University of Dhaka

DU_NE

Emon Khan, Himel Roy, Syed Waki As Sami

2025-01-17

Contest (1)

template.cpp
17 lines
```cpp
#include <bits/stdc++.h>
using namespace std;

#define rep(i, a, b) for(int i = a; i<(b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
#ifdef ONPC
  cerr << endl << "finished in " << clock() * 1.0 /
      CLOCKS_PER_SEC << " sec" << endl;
#endif
}
```

.bashrc
3 lines
```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc
6 lines
```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on    |  im jk <esc>   |  im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
  \| md5sum \| cut -c-6
```

hash.sh
3 lines
```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

stress.sh
10 lines
```bash
#!/bin/bash
[ "$#" -ne 3 ] && echo "Usage: $0 test_file brute_file
    mycode_file" && exit 1
g++ -O2 $1 -o test && g++ -O2 $2 -o brute && g++ -O2 $3 -o
    mycode
for i in {1..10000}; do
  ./test > tests.txt
  ./brute < tests.txt > correct.txt
  ./mycode < tests.txt > myans.txt
  diff -q correct.txt myans.txt >/dev/null || { echo -e "\e[31
      mTest $i: WA\e[0m"; cat tests.txt; break; }
  echo -e "\e[32mTest $i: AC\e[0m"
done
```

interactiveStress.py
19 lines
```python
import subprocess, random
def generate_permutation(n): return random.sample(range(1, n +
    1), n)
def handle_queries(hidden, n, max_q=6666):
    process = subprocess.Popen(["./solve"], stdin=subprocess.
        PIPE, stdout=subprocess.PIPE, text=True)
    process.stdin.write(f"{n}\n"); process.stdin.flush()
    for _ in range(max_q):
        query = process.stdout.readline().strip().split()
        if query[0] == "1":
            print("Correct!" if list(map(int, query[1:])) ==
                hidden else "Wrong!")
            break
        matches = sum(p == h for p, h in zip(map(int, query
            [1:]), hidden))
        process.stdin.write(f"{matches}\n"); process.stdin.
            flush()
    else: print("Query limit exceeded!")
    process.terminate()

n = 1000
hidden_permutation = generate_permutation(n)
print("Hidden permutation:", hidden_permutation)
handle_queries(hidden_permutation, n)
```

makefile
10 lines
```makefile
# runs by make run file=filename, use *tab*
CC = g++
CFLAGS = -fsanitize=address -std=c++17 -Wall -Wextra -Wshadow -
    DONPC -O2
all:
%: %.cpp
    $(CC) $(CFLAGS) -o "$@" "$<"
run: $(file)
    ./$(file)
clean:
    find . -type f -executable -delete
```

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \dfrac{ed - bf}{ad - bc} \\ y = \dfrac{af - ec}{ad - bc} \end{matrix}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v + w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w.

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \text{atan2}(b, a)$.

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \dfrac{a + b + c}{2}$

Area: $A = \sqrt{p(p-a)(p-b)(p-c)}$

Circumradius: $R = \dfrac{abc}{4A}$

Inradius: $r = \dfrac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b+c} \right)^2 \right]}$$

Law of sines: $\dfrac{\sin \alpha}{a} = \dfrac{\sin \beta}{b} = \dfrac{\sin \gamma}{c} = \dfrac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents: $\dfrac{a+b}{a-b} = \dfrac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$
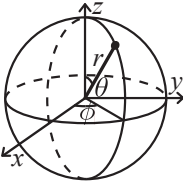
2.4.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.4.3 Spherical coordinates

$$x = r \sin \theta \cos \phi \qquad r = \sqrt{x^2 + y^2 + z^2}$$
$$y = r \sin \theta \sin \phi \qquad \theta = \text{acos}(z / \sqrt{x^2 + y^2 + z^2})$$
$$z = r \cos \theta \qquad \phi = \text{atan2}(y, x)$$

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1 - x^2}} \qquad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1 - x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x \qquad \frac{d}{dx} \arctan x = \frac{1}{1 + x^2}$$

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \text{erf}(x) \qquad \int x e^{ax} dx = \frac{e^{ax}}{a^2}(ax - 1)$$

Integration by parts:

$$\int_a^b f(x) g(x) dx = [F(x) g(x)]_a^b - \int_a^b F(x) g'(x) dx$$

2.6 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2 + 3n - 1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$

$$\sqrt{1 + x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

2.8.1 Discrete distributions
2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\text{U}(a, b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\text{Exp}(\lambda), \lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

Data structures (3)

OrderStatisticTree.h
Description: ...
Time: $\mathcal{O}(\log N)$
<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp>   d41d8c, 14 lines

```cpp
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type, less<int>, rb_tree_tag
    , tree_order_statistics_node_update>
#define ordered_pair_set tree<pair<int, int>, null_type, less<
    pair<int, int>>, rb_tree_tag,
    tree_order_statistics_node_update>
ordered_set os;
// Example using ordered_set
os.insert(5);os.insert(1);os.insert(10);os.insert(3);
cout << "2nd smallest element: " << *os.find_by_order(2) <<
    endl;  // Output: 5
cout << "Elements less than 6: " << os.order_of_key(6) << endl;
        // Output: 3
// Example using ordered_pair_set
ordered_pair_set ops;
ops.insert({1, 100});ops.insert({2, 200});ops.insert({1, 150});
    ops.insert({3, 250});
cout << "1st smallest pair: (" << ops.find_by_order(0)->first
    << ", " << ops.find_by_order(0)->second << ")" << endl;
    // Output: (1, 100)
cout << "Pairs less than (2, 150): " << ops.order_of_key({2,
    150}) << endl;  // Output: 2
```

HashMap.h
Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).
<bits/extc++.h>   d41d8c, 6 lines

```cpp
struct chash {
    const uint64_t C = uint64_t(4e18 * acos(0)) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x * C)
        ; }
};

__gnu_pbds::gp_hash_table<ll, int, chash> h;
```

SegmentTree.h
Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.
Time: $\mathcal{O}(\log N)$
     d41d8c, 113 lines

```cpp
struct Segtree {
    // 0 base indexing
    int n;
    vector<ll> tree;
    ll merge(ll x, ll y) {
        return x + y;
    }
    void build(vector<ll> &a, int node, int l, int r) {
        if(l == r) {
            tree[node] = a[l];
            return;
        }
        int mid = l + ((r - l) >> 1);
        build(a, (node << 1)+1, l, mid);
        build(a, (node << 1)+2, mid+1, r);
        tree[node] = merge(tree[(node << 1)+1], tree[(node <<
            1)+2]);
    }
    void update(int i, ll value, int node, int l, int r) {
        if(l == i && r == i) {
            tree[node] = value;
            return;
        }
        int mid = l + ((r-l) >> 1);
        if(i <= mid)update(i, value, (node << 1)+1, l, mid);
        else update(i, value, (node << 1)+2, mid+1, r);
        tree[node] = merge(tree[(node << 1)+1], tree[(node <<
            1)+2]);
    }
    void update(int i, int value) {
        update(i, value, 0, 0, n-1);
    }
    ll query(int i, int j, int node, int l, int r) {
        if(l > j || r < i) return 0;
        if(l >= i && r <= j)return tree[node];
        int mid = l + ((r - l) >> 1);
        return merge(query(i, j, (node << 1)+1, l, mid), query(
            i, j, (node << 1)+2, mid+1, r));
    }
    ll query(int i, int j) {
        return query(i, j, 0, 0, n-1);
    }
    void init(vector<ll> &a, int _n) {
        n = _n;
        int size = 1;
        while(size < n) size = size << 1;
        tree.resize((size << 1)-1);
        build(a, 0, 0, n-1);
    }
} st;
struct Segtree {
    // 0 base indexing
    int n;
    vector<ll> tree, lazy;

    ll merge(ll x, ll y) {
        return x + y;
    }
    void push(int node, int l, int r) {
        int a = (node << 1)+1, b = (node << 1)+2;
        int mid = l + ((r-l) >> 1);
        tree[a]+=(mid-l+1)*lazy[node], tree[b]+=(r-(mid+1)+1)*
            lazy[node];
        lazy[a]+=lazy[node], lazy[b]+=lazy[node];
        lazy[node] = 0;
    }
    void build(vector<ll> &a, int node, int l, int r) {
        if(l == r) {
            tree[node] = a[l];
            return;
        }
        int mid = l + ((r-l) >> 1);
        build(a, (node << 1)+1, l, mid);
        build(a, (node << 1)+2, mid+1, r);
        tree[node] = merge(tree[(node << 1)+1], tree[(node <<
            1)+2]);
    }
    void build(vector<ll> &a) {
        build(a, 0, 0, n-1);
    }
    void update(int i, int j, ll value, int node, int l, int r)
        {
        if(l > j || r < i)return;
        if(l >= i && r <= j) {
            lazy[node]+=value;
            tree[node]+=(r-l+1)*value;
            return;
        }
        if(lazy[node])push(node, l, r);
        int mid = l + ((r-l) >> 1);
        update(i, j, value, (node << 1)+1, l, mid);
        update(i, j, value, (node << 1)+2, mid+1, r);
        tree[node] = merge(tree[(node << 1)+1], tree[(node <<
            1)+2]);
    }
    void update(int i, int j, ll value) {
        update(i, j, value, 0, 0, n-1);
    }
    ll query(int i, int j, int node, int l, int r) {
        if(l > j || r < i)
            return 0;
        if(l >= i && r <= j)
            return tree[node];
        if(lazy[node]) push(node, l, r);
        int mid = l + ((r-l) >> 1);
        return merge(query(i, j, (node << 1)+1, l, mid), query(
            i, j, (node << 1)+2, mid+1, r));
    }
    ll query(int i, int j) {
        return query(i, j, 0, 0, n-1);
    }
    void init(vector<ll> &a, int _n) {
        n = _n;
        int size = 1;
        while(size < n) size = size << 1;
        tree.resize((size << 1)-1);
        lazy.assign((size << 1)-1, 0);
        build(a, 0, 0, n-1);
    }
} st;
```

LazySegmentTree.h
Description: Segment tree with lazy propagation
Usage: update(1, 0, n - 1, ql, qr, val), query(1, 0, n - 1, ql, qr)
Time: $\mathcal{O}(\log N)$
     d41d8c, 66 lines

```cpp
struct Segtree {
    // 0 base indexing
    int n;
    vector<ll> tree, lazy;

    ll merge(ll x, ll y) {
        return x + y;
    }
    void push(int node, int l, int r) {
        int a = (node << 1)+1, b = (node << 1)+2;
        int mid = l + ((r-l) >> 1);
        tree[a]+=(mid-l+1)*lazy[node], tree[b]+=(r-(mid+1)+1)*
            lazy[node];
        lazy[a]+=lazy[node], lazy[b]+=lazy[node];
        lazy[node] = 0;
```

```cpp
        }
        void build(vector<ll> &a, int node, int l, int r) {
            if(l == r) {
                tree[node] = a[l];
                return;
            }
            int mid = l + ((r-l) >> 1);
            build(a, (node << 1)+1, l, mid);
            build(a, (node << 1)+2, mid+1, r);
            tree[node] = merge(tree[(node << 1)+1], tree[(node <<
                1)+2]);
        }
        void build(vector<ll> &a) {
            build(a, 0, 0, n-1);
        }
        void update(int i, int j, ll value, int node, int l, int r)
            {
            if(l > j || r < i)return;
            if(l >= i && r <= j) {
                lazy[node]+=value;
                tree[node]+=(r-l+1)*value;
                return;
            }
            if(lazy[node])push(node, l, r);
            int mid = l + ((r-l) >> 1);
            update(i, j, value, (node << 1)+1, l, mid);
            update(i, j, value, (node << 1)+2, mid+1, r);
            tree[node] = merge(tree[(node << 1)+1], tree[(node <<
                1)+2]);
        }
        void update(int i, int j, ll value) {
            update(i, j, value, 0, 0, n-1);
        }
        ll query(int i, int j, int node, int l, int r) {
            if(l > j || r < i)
                return 0;
            if(l >= i && r <= j)
                return tree[node];

            if(lazy[node]) push(node, l, r);
            int mid = l + ((r-l) >> 1);
            return merge(query(i, j, (node << 1)+1, l, mid), query(
                i, j, (node << 1)+2, mid+1, r));
        }
        ll query(int i, int j) {
            return query(i, j, 0, 0, n-1);
        }
        void init(vector<ll> &a, int _n) {
            n = _n;
            int size = 1;
            while(size < n) size = size << 1;
            tree.resize((size << 1)-1);
            lazy.assign((size << 1)-1, 0);
            build(a, 0, 0, n-1);
        }
} st;
```

PersistentSegtree.h
Description: PresistentSegment Tree

```cpp
struct persistentSegtree {
    // 0 base indexing
    ll data;
    persistentSegtree *left, *right;

    ll merge(ll x, ll y) {
        return x + y;
    }
    void build(vector<ll> &a, int l, int r) {
```

```cpp
            if(l == r) {
                data = a[l];
                return;
            }
            int mid = l + ((r - l) >> 1);
            left = new persistentSegtree();
            right = new persistentSegtree();
            left->build(a, l, mid);
            right->build(a, mid+1, r);
            data = merge(left->data, right->data);
        }
        persistentSegtree* update(int i, ll value, int l, int r) {
            if(l > i || r < i) return this;
            if(l == i && r == i) {
                persistentSegtree *rslt = new persistentSegtree();
                rslt->data = value;
                return rslt;
            }
            int mid = l + ((r-l) >> 1);
            persistentSegtree *rslt = new persistentSegtree();

            rslt->left = left->update(i, value, l, mid);
            rslt->right = right->update(i, value, mid+1, r);
            rslt->data = merge(rslt->left->data, rslt->right->data)
                ;

            return rslt;
        }
        ll query(int i, int j, int l, int r) {
            if(l > j || r < i) return 0;
            if(l >= i && r <= j)return data;
            int mid = l + ((r - l) >> 1);
            return merge(left->query(i, j, l, mid), right->query(i,
                j, mid+1, r));
        }
} *roots[N];
int main() {// Idea from Mahmudul Yeamim
    int tt = 1;
    while(tt--) {
        int n, q, k = 0;
        cin >> n >> q;
        vector<ll> a(n);
        for(int i = 0; i < n; i++) {
            cin >> a[i];
        }
        roots[0] = new persistentSegtree();
        roots[k++]->build(a, 0, n-1);
        while(q--) {
            int type;
            cin >> type;
            if(type == 1) {
                int _k, i;
                ll x;
                cin >> _k >> i >> x;
                --_k;
                roots[_k] = roots[_k]->update(--i, x, 0, n-1);
            }else if(type == 2) {
                int _k, i, j;
                cin >> _k >> i >> j;
                cout << roots[--_k]->query(--i, --j, 0, n-1) <<
                    "\n";
            }else {
                int _k;
                cin >> _k;
                roots[k++] = roots[--_k];
            }
        }
    }
    return 0;
}
```

```cpp
}
```

UnionFind.h
Description: Disjoint-set data structure.
Time: $\mathcal{O}\left(\alpha(N)\right)$

```cpp
void make_set(int v) {
    parent[v] = v;
    Size[v] = 1;
}
int find_set(int v) {
    if (v == parent[v])return v;
    return parent[v] = find_set(parent[v]);
}
void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if(Size[a] < Size[b])swap(a, b);
        parent[b] = a;
        Size[a]+=Size[b];
    }
}
```

UnionFindRollback.h
Description: 2D prefix with update

2DPrefix.h
Description: 2D prefix with update
Usage: SubMatrix<int> m(matrix);
m.sum(0, 0, 2, 2); // top left 4 elements
Time: $\mathcal{O}\left(N^2 + Q\right)$

```cpp
void update(vector<vector<ll>>& grid, int x1, int y1, int x2,
    int y2, int val) {
    grid[x1][y1] += val;
    if (x2 + 1 < n) grid[x2 + 1][y1] -= val;
    if (y2 + 1 < m) grid[x1][y2 + 1] -= val;
    if (x2 + 1 < n && y2 + 1 < m) grid[x2 + 1][y2 + 1] += val;
}
vector<vector<ll>> calculate(vector<vector<ll>> &grid) {
    vector<vector<ll>> ans(n, vector<ll>(m, 0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            ans[i][j] = grid[i][j];
            if(i > 0) ans[i][j] += ans[i - 1][j];
            if(j > 0) ans[i][j] += ans[i][j - 1];
            if(i > 0 && j > 0) ans[i][j] -= ans[i - 1][j - 1];
        }
    }
    return ans;
}
template<class T> struct SubMatrix {
    vector<vector<T>> p;
    SubMatrix(const vector<vector<T>>& v) {
        int R = v.size(), C = v[0].size();
        p.assign(R + 1, vector<T>(C + 1, 0));

        for (int r = 0; r < R; ++r) {
            for (int c = 0; c < C; ++c) {
                p[r + 1][c + 1] = v[r][c] + p[r][c + 1] + p[r +
                    1][c] - p[r][c];
            }
        }
    }
    T sum(int u, int l, int d, int r) {
        return p[d + 1][r + 1] - p[u][r + 1] - p[d + 1][l] + p[
            u][l];
```

```
        }
};
```

Matrix.h
Description: Basic operations on square matrices.
Usage: Matrix<int, 3, 3> A;
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
vector<int> vec = {1,2,3};
vec = (A^N) * vec;
d41d8c, 34 lines

```cpp
template<class T, int N, int M> struct Matrix {
    typedef Matrix Mx;
    array<array<T, M>, N> d{};
    // Matrix multiplication
    template<int P>
    Matrix<T, N, P> operator*(const Matrix<T, M, P>& m) const {
        Matrix<T, N, P> a;
        for (int i = 0; i < N; i++)
            for (int j = 0; j < P; j++)
                for (int k = 0; k < M; k++)
                    a.d[i][j] += d[i][k] * m.d[k][j];
        return a;
    }
    // Matrix-vector multiplication
    vector<T> operator*(const vector<T>& vec) const {
        vector<T> ret(N, 0);
        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                ret[i] += d[i][j] * vec[j];
        return ret;
    }
    // Matrix exponentiation
    Matrix<T, N, N> operator^(ll p) const {
        static_assert(N == M); assert(p >= 0);
        Matrix<T, N, N> a, b(*this);
        for (int i = 0; i < N; i++) a.d[i][i] = 1; // Identity
            matrix
        while (p) {
            if (p & 1) a = a * b;
            b = b * b;
            p >>= 1;
        }
        return a;
    }
};
```

CHT.h
Description: Container where you can add lines of the form kx+m, and query minimum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$
d41d8c, 38 lines

```cpp
struct Line {
    // m = slope, c = intercept
    ll m, c;
    Line(ll a, ll b) : m(a), c(b) {}
};
struct CHT {
    // SayeefMahmud
    vector<Line> lines;

    bool bad(Line l1, Line l2, Line l3) {
        __int128 a = (__int128)(l2.c - l1.c) * (l2.m - l3.m);
        __int128 b = (__int128)(l3.c - l2.c) * (l1.m - l2.m);
        return a >= b;
    }
    void add(Line line) {
        lines.push_back(line);
        int sz = lines.size();
```

```cpp
        while (sz >= 3 && bad(lines[sz - 3], lines[sz - 2],
            lines[sz - 1])) {
            lines.erase(lines.end() - 2);
            sz--;
        }
    }
    ll query(ll x) {
        int l = 0, r = lines.size() - 1;
        ll ans = LLONG_MAX;
        while (l <= r) {
            int mid1 = l + (r - l) / 3;
            int mid2 = r - (r - l) / 3;
            ans = min(ans, min(lines[mid1].m * x + lines[mid1].
                c, lines[mid2].m * x + lines[mid2].c));
            if (lines[mid1].m * x + lines[mid1].c <= lines[mid2
                ].m * x + lines[mid2].c) {
                r = mid2 - 1;
            } else {
                l = mid1 + 1;
            }
        }
        return ans;
    }
};
```

Treap.h
Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$

FenwickTree.h
Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
d41d8c, 26 lines

```cpp
struct FenwickTree {
    // 0 base indexing
    vector<int> bit;
    int n;
    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }
    FenwickTree(vector<int> const &a) : FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size(); i++)
            add(i, a[i]);
    }
    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }
    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
    void add(int idx, int delta) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }
};
```

FenwickTree2d.h
Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
d41d8c, 36 lines

```cpp
struct FenwickTree2D {
```

```cpp
    // 0 base indexing
    vector<vector<int>> bit;
    int n, m;
    FenwickTree2D(int n, int m) {
        this->n = n;
        this->m = m;
        bit.assign(n, vector<int>(m, 0));
    }
    FenwickTree2D(vector<vector<int>>& matrix) : FenwickTree2D(
        matrix.size(), matrix[0].size()) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                add(i, j, matrix[i][j]);
            }
        }
    }
    int sum(int x, int y) {
        int ret = 0;
        for (int i = x; i >= 0; i = (i & (i + 1)) - 1) {
            for (int j = y; j >= 0; j = (j & (j + 1)) - 1) {
                ret += bit[i][j];
            }
        }
        return ret;
    }
    int sum(int x1, int y1, int x2, int y2) {
        return sum(x2, y2) - sum(x2, y1 - 1) - sum(x1 - 1, y2)
            + sum(x1 - 1, y1 - 1);
    }
    void add(int x, int y, int delta) {
        for (int i = x; i < n; i = i | (i + 1)) {
            for (int j = y; j < m; j = j | (j + 1)) {
                bit[i][j] += delta;
            }
        }
    }
};
```

RMQ.h
Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: RMQ rmq(values);
rmq.query(inclusive, exclusive);
Time: $\mathcal{O}(|V| \log |V| + Q)$
d41d8c, 26 lines

```cpp
struct RMQ {
    // 0-base indexing
    int n, logN;
    vector<vector<int>> st;
    vector<int> lg;

    void init(const vector<int>& array) {
        n = array.size();
        logN = ceil(log2(n));
        st.resize(logN, vector<int>(n));
        lg.resize(n + 1);
        lg[1] = 0;
        for (int i = 2; i <= n; i++)
            lg[i] = lg[i / 2] + 1;
        copy(array.begin(), array.end(), st[0].begin());
        for (int i = 1; i < logN; i++) {
            for (int j = 0; j + (1 << i) <= n; j++) {
                st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i -
                    1))]);
            }
        }
    }
    int query(int L, int R) {
        int i = lg[R - L + 1];
```

```
        return min(st[i][L], st[i][R - (1 << i) + 1]);
    }
} ST;
```

MoQueries.h
Description: ...

d41d8c, 48 lines

```
// 0-base indexing
void add(int x) {
    if(!freq[x]) distinct++;
    freq[x]++;
}
void remove(int x) {
    freq[x]--;
    if(!freq[x]) distinct--;
}
void adjust(int &curr_l, int &curr_r, int L, int R) {
    while(curr_l > L) {
        curr_l--;
        add(a[curr_l]);
    }
    while(curr_r < R) {
        curr_r++;
        add(a[curr_r]);
    }
    while(curr_l < L) {
        remove(a[curr_l]);
        curr_l++;
    }
    while(curr_r > R) {
        remove(a[curr_r]);
        curr_r--;
    }
}
void solve(vector<array<int, 3>> &queries) {
    // const int BLOCK_SIZE = sqrt(queries.size()) + 1;
    const int BLOCK_SIZE = 555;
    sort(queries.begin(), queries.end(), [&](const array<int,
        3>& a, const array<int, 3>& b) {
        int blockA = a[0] / BLOCK_SIZE;
        int blockB = b[0] / BLOCK_SIZE;
        if (blockA != blockB)
            return blockA < blockB;
        return a[1] < b[1];
    });
    auto[L, R, id] = queries[0];
    int curr_l = L, curr_r = L;
    distinct = 1;
    freq[a[curr_l]]++;
    vector<int> ans(queries.size());
    for(auto [L, R, id] : queries) {
        adjust(curr_l, curr_r, L, R);
        ans[id] = distinct;
    }
    for(auto x : ans) cout << x << "\n";
}
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h
d41d8c, 17 lines

```
struct Poly {
    vector<double> a;
    double operator()(double x) const {
        double val = 0;
        for (int i = sz(a); i--;) (val *= x) += a[i];
```

```
        return val;
    }
    void diff() {
        rep(i,1,sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    }
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
        a.pop_back();
    }
};
```

PolyRoots.h
Description: Finds the real roots to a polynomial.
Usage: polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0
Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

"Polynomial.h"   d41d8c, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    rep(i,0,sz(dr)-1) {
        double l = dr[i], h = dr[i+1];
        bool sign = p(l) > 0;
        if (sign ^ (p(h) > 0)) {
            rep(it,0,60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m;
                else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    }
    return ret;
}
```

PolyInterpolate.h
Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

d41d8c, 13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1;
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    }
    return res;
}
```

BerlekampMassey.h
Description: Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.
Usage: berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}

Time: $\mathcal{O}\left(N^2\right)$

"../number-theory/ModPow.h"   d41d8c, 18 lines

```
vector<ll> berlekampMassey(vector<ll> s) {
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;
    ll b = 1;
    rep(i,0,n) { ++m;
        ll d = s[i] % mod;
        rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod;
        rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }
    C.resize(L + 1); C.erase(C.begin());
    for (ll& x : C) x = (mod - x) % mod;
    return C;
}
```

LinearRecurrence.h
Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \ldots \geq n-1]$ and $tr[0 \ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number
Time: $\mathcal{O}\left(n^2 \log k\right)$

d41d8c, 22 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);
    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
        return res;
    };
    Poly pol(n + 1), e(pol);
    pol[0] = e[1] = 1;
    for (++k; k; k /= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    }
    ll res = 0;
    rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
    return res;
}
```

4.2 Optimization

GoldenSectionSearch.h
Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is eps. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: double func(double x) { return 4+x+.3*x*x; }
double xmin = gss(-1000,1000,func);
Time: $\mathcal{O}(\log((b-a)/\epsilon))$

d41d8c, 14 lines

```
double gss(double a, double b, double (*f)(double)) {
    double r = (sqrt(5)-1)/2, eps = 1e-7;
    double x1 = b - r*(b-a), x2 = a + r*(b-a);
    double f1 = f(x1), f2 = f(x2);
    while (b-a > eps)
        if (f1 < f2) { //change to > to find maximum
```

```cpp
        b = x2; x2 = x1; f2 = f1;
        x1 = b - r*(b-a); f1 = f(x1);
    } else {
        a = x1; x1 = x2; f1 = f2;
        x2 = a + r*(b-a); f2 = f(x2);
    }
    return a;
}
```

HillClimbing.h
Description: Poor man's optimization for unimodal functions.
d41d8c, 14 lines

```cpp
typedef array<double, 2> P;

template<class F> pair<double, P> hillClimb(P start, F f) {
    pair<double, P> cur(f(start), start);
    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
        rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        }
    }
    return cur;
}
```

Integrate.h
Description: Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4, although in practice you will want to verify that the result is stable to desired precision when epsilon changes.
d41d8c, 7 lines

```cpp
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
    double h = (b - a) / 2 / n, v = f(a) + f(b);
    rep(i,1,n*2)
        v += f(a + i*h) * (i&1 ? 4 : 2);
    return v * h / 3;
}
```

IntegrateAdaptive.h
Description: Fast integration using an adaptive Simpson's rule.
Usage: double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [&](double y) {
return quad(-1, 1, [&](double z) {
return x*x + y*y + z*z < 1; });});});
d41d8c, 15 lines

```cpp
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
    d c = (a + b) / 2;
    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
    return rec(f, a, b, eps, S(a, b));
}
```

Simplex.h
Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: vvd A = {{1,-1}, {-1,1}, {-1,-2}};
vd b = {1,1,-4}, c = {-1,-1}, x;
T val = LPSolver(A, b, c).solve(x);
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.
d41d8c, 62 lines

```cpp
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
struct LPSolver {
    int m, n;
    vi N, B;
    vvd D;
    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
            rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
            rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
            rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
            N[n] = -1; D[m+1][n] = 1;
        }
    void pivot(int r, int s) {
        T *a = D[r].data(), inv = 1 / a[s];
        rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
            T *b = D[i].data(), inv2 = b[s] * inv;
            rep(j,0,n+2) b[j] -= a[j] * inv2;
            b[s] = a[s] * inv2;
        }
        rep(j,0,n+2) if (j != s) D[r][j] *= inv;
        rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }
    bool simplex(int phase) {
        int x = m + phase - 1;
        for (;;) {
            int s = -1;
            rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
            if (D[x][s] >= -eps) return true;
            int r = -1;
            rep(i,0,m) {
                if (D[i][s] <= eps) continue;
                if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                             < MP(D[r][n+1] / D[r][s], B[r])) r = i;
            }
            if (r == -1) return false;
            pivot(r, s);
        }
    }
    T solve(vd &x) {
        int r = 0;
        rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
        if (D[r][n+1] < -eps) {
            pivot(r, n);
            if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
            rep(i,0,m) if (B[i] == -1) {
                int s = 0;
                rep(j,1,n+1) ltj(D[i]);
                pivot(i, s);
            }
        }
```

```cpp
        bool ok = simplex(1); x = vd(n);
        rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
        return ok ? D[m][n+1] : inf;
    }
};
```

4.3   Matrices

Determinant.h
Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$
d41d8c, 15 lines

```cpp
double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1;
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
        }
    }
    return res;
}
```

IntDeterminant.h
Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}(N^3)$
d41d8c, 18 lines

```cpp
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}
```

SolveLinear.h
Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}(n^2 m)$
d41d8c, 36 lines

```cpp
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
```

```
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }
  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h
Description: To get all uniquely determined values of x back from Solve-Linear, make the following changes:

"SolveLinear.h"   d41d8c, 7 lines
```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

SolveLinearBinary.h
Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}\left(n^2 m\right)$

d41d8c, 33 lines
```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }
  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
```

```
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h
Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}\left(n^3\right)$

d41d8c, 32 lines
```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }
  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

MatrixExpo.h
Description: Matrix Exponentiation

d41d8c, 33 lines
```
using row = vector<int>;
using matrix = vector<row>;
matrix unit_mat(int n) {
  matrix I(n, row(n));
  for (int i = 0; i < n; ++i){
    I[i][i] = 1;
  }
  return I;
}

matrix mat_mul(matrix a, matrix b) {
  int m = a.size(), n = a[0].size();
  int p = b.size(), q = b[0].size();
  // assert(n==p);
  matrix res(m, row(q));
  for (int i = 0; i < m; ++i){
    for (int j = 0; j < q; ++j){
      for (int k = 0; k < n; ++k){
        res[i][j] = (res[i][j] + a[i][k]*b[k][j]) % mod;
      }
    }
```

```
  }
  return res;
}
matrix mat_exp(matrix a, int p) {
  int m = a.size(), n = a[0].size(); // assert(m==n);
  matrix res = unit_mat(m);
  while (p) {
    if (p&1)  res = mat_mul(a, res);
    a = mat_mul(a, a);
    p >>= 1;
  }
  return res;
}
```

Gauss.h
Description: Gauss

d41d8c, 60 lines
```
ll bigMod (ll a, ll e, ll mod) {
  if (e == -1) e = mod - 2;
  ll ret = 1;
  while (e) {
    if (e & 1) ret = ret * a % mod;
    a = a * a % mod, e >>= 1;
  }
  return ret;
}
pair <int, ld> gaussJordan (int n, int m, ld eq[N][N], ld res[N
    ]) {
  ld det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (fabs(eq[k][j]) > fabs(eq[
        piv][j])) piv = k;
    if (fabs(eq[piv][j]) < EPS) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = -det; det *= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i) {
      ld x = eq[k][j] / eq[i][j];
      for (int l = j; l <= m; ++l) eq[k][l] -= x * eq[i][l];
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[
        pos[i]][m] / eq[pos[i]][i];
  }
  for (int i = 0; i < n; ++i) {
    ld cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += eq[i][j] * res[j];
    if (fabs(cur) > EPS) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanModulo (int n, int m, int eq[N][N],
    int res[N], int mod) {
  int det = 1;
  vector <int> pos(m, -1);
  const ll mod_sq = (ll) mod * mod;
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j] > eq[piv][j]) piv
        = k;
    if (!eq[piv][j]) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = det ? MOD - det : 0; det = (ll) det * eq
        [i][j] % MOD;
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) {
      ll x = eq[k][j] * bigMod(eq[i][j], -1, mod) % mod;
```

```cpp
      for (int l = j; l <= m; ++l) if (eq[i][l]) eq[k][l] = (eq
            [k][l] + mod_sq - x * eq[i][l]) % mod;
      } ++i;
    }
    int free_var = 0;
    for (int i = 0; i < m; ++i) {
      pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[
          pos[i]][m] * bigMod(eq[pos[i]][i], -1, mod) % mod;
    }
    for (int i = 0; i < n; ++i) {
      ll cur = -eq[i][m];
      for (int j = 0; j < m; ++j) cur += (ll) eq[i][j] * res[j],
          cur %= mod;
      if (cur) return make_pair(-1, det);
    }
    return make_pair(free_var, det);
}
```

Tridiagonal.h
Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, \ 1 \le i \le n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, ..., -1, 1\}, \{0, c_1, c_2, \dots, c_n\},$$
$$\{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$ d41d8c, 26 lines

```cpp
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
      const vector<T>& sub, vector<T> b) {
  int n = sz(b); vi tr(n);
  rep(i,0,n-1) {
    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
      b[i+1] -= b[i] * diag[i+1] / super[i];
      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
      diag[i+1] = sub[i]; tr[++i] = 1;
    } else {
      diag[i+1] -= super[i]*sub[i]/diag[i];
      b[i+1] -= b[i]*sub[i]/diag[i];
    }
  }
  for (int i = n; i--;) {
    if (tr[i]) {
      swap(b[i], b[i-1]);
      diag[i-1] = diag[i];
      b[i] /= super[i-1];
    } else {
      b[i] /= diag[i];
      if (i) b[i-1] -= b[i]*super[i-1];
    }
  }
  return b;
}
```

Xorbasis.h
Description: Xor basis   d41d8c, 13 lines

```cpp
int basis[d] = {0};
int sz = 0;
void insertVector(int mask) {
    for (int i = 0; i < d; i++) {
        if ((mask & (1 << i)) == 0) continue;
        if (!basis[i]) {
            basis[i] = mask;
            ++sz;
            return;
        }
        mask ^= basis[i];
    }
}
```

4.4   Fourier transforms

FastFourierTransform.h
Description: Returns coefficient of multiplication of two polynomials.
                                    d41d8c, 46 lines

```cpp
const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a), b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w *
          wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
}
vector<long long> multiply(vector<ll> &a, vector<ll> &b) {
  int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long long) round(z[i].a
      );
  while((int)ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
```

FastFourierTransformMod.h
Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.
Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)
"FastFourierTransform.h"   d41d8c, 22 lines

```cpp
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
    vector<C> L(n), R(n), outl(n), outs(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

NumberTheoreticTransform.h
Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x] g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For arbitrary modulo, see FFTMod. conv(a, b) = c, where $c[x] = \sum a[i]b[x - i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in $[0, \text{mod})$.
Time: $\mathcal{O}(N \log N)$
"../number-theory/ModPow.h"   d41d8c, 35 lines

```cpp
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
        n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
```

```cpp
  ntt(L), ntt(R);
  rep(i,0,n)
    out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h
Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$, where $\oplus$ is one of AND, OR, XOR. The size of a must be a power of two.
Time: $\mathcal{O}(N \log N)$
"" d41d8c, 16 lines

```cpp
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
      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
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  rep(i,0,sz(a)) a[i] *= b[i];
  FST(a, 1); return a;
}
```

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h
Description: Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.
"euclid.h" d41d8c, 18 lines

```cpp
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

ModInverse.h
Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime.
d41d8c, 3 lines

```cpp
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

ModPow.h
d41d8c, 8 lines

```cpp
const ll mod = 1000000007; // faster if const

ll modpow(ll b, ll e, ll mod) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModLog.h
Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod{m}$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.
Time: $\mathcal{O}(\sqrt{m})$
d41d8c, 11 lines

```cpp
ll modLog(ll a, ll b, ll m) {
  ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m)
    A[e * b % m] = j++;
  if (e == b % m) return j;
  if (__gcd(m, e) == __gcd(m, b))
    rep(i,2,n+2) if (A.count(e = e * f % m))
      return n * i - A[e];
  return -1;
}
```

ModSum.h
Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) $= \sum_{i=0}^{\text{to}-1}(ki+c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant.
d41d8c, 14 lines

```cpp
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}
ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h
Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow
d41d8c, 11 lines

```cpp
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
```

ModSqrt.h
Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).

Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}(\log p)$ for most p
"ModPow.h" d41d8c, 24 lines

```cpp
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

FastEratosthenes.h
Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s
d41d8c, 20 lines

```cpp
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
  const int S = (int)round(sqrt(LIM)), R = LIM / 2;
  vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1));
  vector<pii> cp;
  for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
    cp.push_back({i, i * i / 2});
    for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
  }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto &[p, idx] : cp)
      for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
    rep(i,0,min(S, R - L))
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

MillerRabin.h
Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \bmod c$.
"ModMulLL.h" d41d8c, 12 lines

```cpp
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
    s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {   // ^ count trailing zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
```

```
    }
    return 1;
}
```

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h"   d41d8c, 18 lines

```cpp
ull pollard(ull n) {
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    auto f = [&](ull x) { return modmul(x, x, n) + i; };
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}
vector<ull> factor(ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r));
    return l;
}
```

5.3 Divisibility

euclid.h

Description: Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in __gcd instead. If a and b are coprime, then x is the inverse of $a \pmod{b}$.

d41d8c, 5 lines

```cpp
ll euclid(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d;
}
```

5.3.1 Chinese Remainder Theorem

Let $m = m_1 \cdot m_2 \cdots m_k$, where m_i are pairwise coprime. In addition to m_i, we are also given a system of congruences

$$
\begin{cases}
a & \equiv & a_1 \pmod{m_1} \\
a & \equiv & a_2 \pmod{m_2} \\
& \vdots & \\
a & \equiv & a_k \pmod{m_k}
\end{cases}
$$

where a_i are some given constants. CRT will give the unique solution modulo m.

CRT.h

Description: Chinese Remainder Theorem.
crt(a, m, b, n) computes x such that $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.

Time: $\log(n)$

"euclid.h"   d41d8c, 7 lines

```cpp
ll crt(ll a, ll m, ll b, ll n) {
    if (n > m) swap(a, b), swap(m, n);
    ll x, y, g = euclid(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
```

```cpp
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m*n/g : x;
}
```

CRT2.h

Description: Chinese Remainder Theorem.

Time: $\mathcal{O}(n)$ here n is the number of congruences.

d41d8c, 17 lines

```cpp
struct Congruence {
    ll a, m;
};
ll CRT(vector<Congruence> const &congruences) {
    ll M = 1;
    for (auto const &congruence : congruences) {
        M *= congruence.m;
    }
    ll solution = 0;
    for (auto const &congruence : congruences) {
        ll a_i = congruence.a;
        ll M_i = M / congruence.m;
        ll N_i = mod_inv(M_i, congruence.m);
        solution = (solution + a_i * M_i % M * N_i) % M;
    }
    return solution;
}
```

5.3.2 Bézout's identity

For $a \ne$, $b \ne 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$
\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)} \right), \quad k \in \mathbb{Z}
$$

Diophantine.h

Description: Provides any solution of $ax + by = c$

Time: $\mathcal{O}(\log(n))$

"euclid.h"   d41d8c, 8 lines

```cpp
bool find_any_solution(int a, int b, int c, int &x0, int &y0,
        int &g) {
    g = euclid(abs(a), abs(b), x0, y0);
    if (c % g) return false;
    x0 *= c / g, y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

phiFunction.h

Description: Euler's ϕ function is defined as $\phi(n) := \#$ of positive integers $\le n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$. $\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.

Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p} \; \forall a$.

Time: $\mathcal{O}(\log \log n)$ and $\mathcal{O}(\sqrt{n})$ for the second version.

d41d8c, 21 lines

```cpp
const int LIM = 5000000;
int phis[LIM];
```

```cpp
void calculatePhi() {
    rep(i, 0, LIM) phis[i] = i & 1 ? i : i / 2;
    for (int i = 3; i < LIM; i += 2)
        if (phis[i] == i)
            for (int j = i; j < LIM; j += i)
                phis[j] -= phis[j] / i;
}
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            result -= result / i;
        }
    }
    if (n > 1) result -= result / n;
    return result;
}
```

5.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \ge 0$, finds the closest rational approximation p/q with $p, q \le N$. It will obey $|p/q - x| \le 1/qN$.

For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.

Time: $\mathcal{O}(\log N)$

d41d8c, 21 lines

```cpp
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
    for (;;) {
        ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
            a = (ll)floor(y), b = min(a, lim),
            NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If b > a/2, we have a semi-convergent that gives us a
            // better approximation; if b = a/2, we *may* have one.
            // Return {P, Q} here for a more canonical approximation.
            return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
                make_pair(NP, NQ) : make_pair(P, Q);
        }
        if (abs(y = 1/(y - (d)a)) > 3*N) {
            return {NP, NQ};
        }
        LP = P; P = NP;
        LQ = Q; Q = NQ;
    }
}
```

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \le N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.

Usage: fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}

Time: $\mathcal{O}(\log(N))$

d41d8c, 25 lines

```cpp
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >>= si) {
            adv += step;
```

```
        Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
        if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
            adv -= step; si = 2;
        }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
  }
  return dir ? hi : lo;
}
```

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \ \ b = k \cdot (2mn), \ \ c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

5.6 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^{\times}$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.7 Fibonacchi

Fibonacci numbers are defined by
$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$. Again, $F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} \approx \frac{\phi^n}{\sqrt{5}}$, where $\phi = \frac{1+\sqrt{5}}{2}$ and $\hat{\phi} = \frac{1-\sqrt{5}}{2}$. Some important properties of Fibonacci numbers:

$$F_{n-1}F_{n+1} - F_n^2 = (-1)^n \qquad F_{n+k} = F_{k-1}F_n + F_k F_{n+1}$$
$$F_{2n} = F_n(F_{n-1} + F_{n+1}) \qquad F_{2n+1} = F_n^2 + F_{n+1}^2$$
$$n|m \Leftrightarrow F_n|F_m \qquad \gcd(F_m, F_n) = F_{\gcd(m,n)}$$

Fibonacchi.h
Description: Fast doubling Fibonacci algorithm. Returns F(n) and F(n+1).
Time: $\mathcal{O}(\log n)$
d41d8c, 11 lines

```cpp
pair<int, int> fib(int n) {
  if (n == 0)
    return {0, 1};
  auto p = fib(n >> 1);
  int c = p.first * (2 * p.second - p.first);
  int d = p.first * p.first + p.second * p.second;
  if (n & 1)
    return {d, c + d};
  else
    return {c, d};
}
```

5.8 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

5.9 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)

$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$

$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

n	11	12	13	14	15	16	17
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14

n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

IntPerm.h
Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$
d41d8c, 6 lines

```cpp
int permToInt(vi& v) {
  int use = 0, i = 0, r = 0;
  for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
    use |= 1 << x;                    // (note: minus, not ~!)
  return r;
}
```

6.1.2 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

6.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

6.1.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

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

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k).$$

6.2 Partitions and subsets

6.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

6.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

6.2.3 Binomials

multinomial.h
Description: Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$.
d41d8c, 5 lines

```cpp
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i]) c = c * ++m / (j+1);
  return c;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able).
$B[0, \ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_{m}^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m)$$

$$\approx \int_{m}^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k), \ c(0,0) = 1$$
$$\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

6.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$$

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$$

6.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

6.3.6 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$

6.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \ C_{n+1} = \frac{2(2n+1)}{n+2}C_n, \ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Graph (7)

7.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max |w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$ d41d8c, 64 lines

```cpp
void BellmanFord(int st, int n) {
    vector<ll> dist(n+1, INF);
    vector<int> parent(n+1, -1);
    dist[st] = 0;
    for (int i = 0; i < n-1; i++) {
        bool any = false;
        for (auto[u, v, cost] : edges)
            if (dist[u] < INF)
                if (dist[v] > dist[u] + cost) {
                    dist[v] = dist[u] + cost;
                    parent[v] = u;
                    any = true;
                }
        if (!any)
            break;
    }
    if (dist[n] == INF)
        cout << "-1\n";
    else {
        vector<int> path;
        for (int cur = n; cur != -1; cur = parent[cur])
            path.push_back(cur);
        reverse(path.begin(), path.end());
        for (int u : path)
            cout << u << ' ';
    }
}

void BellmanFord(int s, int n) {
    vector<ll> dist(n+1, 0);// No need to init INF here because
        there can be a negative cycle where you can't reach
        from node 1
                    // and the Graph is not necessarily
                        connected
                    // Our concern is about to find
                        negetive cycle not shortest
                        distance
```

```cpp
    vector<int> parent(n+1, -1);
    dist[s] = 0;
    int flag;
    for (int i = 0; i < n; i++) {
        flag = -1;
        for (auto[u, v, cost] : edges) {
            if (dist[u] + cost < dist[v]) {
                dist[v] = dist[u] + cost;
                parent[v] = u;
                flag = v;
            }
        }
    }
    if (flag == -1)
        cout << "NO\n";
    else {
        int y = flag;
        for (int i = 0; i < n; ++i)
            y = parent[y];

        vector<int> path;
        for (int cur = y;; cur = parent[cur]) {
            path.push_back(cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse(path.begin(), path.end());
        cout << "YES\n";
        for (int u : path)
            cout << u << ' ';
    }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] = \text{inf}$ if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, inf if no path, or -inf if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$ d41d8c, 19 lines

```cpp
void init() {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            d[i][j] = INF;
        }
        d[i][i] = 0;
    }
}
void floydWarshall() {
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (d[i][k] < INF && d[k][j] < INF) {
                    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
                }
            }
        }
    }
}
```

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Time: $\mathcal{O}(|V| + |E|)$ d41d8c, 8 lines

```cpp
vi topoSort(const vector<vi>& gr) {
```

```
  vi indeg(sz(gr)), q;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push_back(i);
  rep(j,0,sz(q)) for (int x : gr[q[j]])
    if (--indeg[x] == 0) q.push_back(x);
  return q;
}
```

7.2 Network flow

PushRelabel.h
Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

d41d8c, 45 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }
  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);
    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h
Description: Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(FE\log(V)\right)$ where F is max flow. $\mathcal{O}\left(VE\right)$ for setpi.

d41d8c, 76 lines

```
const int N = 500;
```

```
vector<int> adj[N+1];
int capacity[N+1][N+1];

int bfs(int s, int d, int n, vector<int> &parent) {
    parent.assign(n+1, -1);
    parent[s] = 0;
    queue<pair<int, int>> q;
    q.push({s, INT_MAX});
    while (!q.empty()) {
        int u = q.front().first;
        int f =  q.front().second;
        q.pop();
        for(auto v : adj[u]) {
            if(parent[v] == -1 && capacity[u][v]) {
                parent[v] = u;
                int n_f = min(f, capacity[u][v]);
                if(v == d)return n_f;
                q.push({v, n_f});
            }
        }
    }
    return 0;
}
int max_flow(int s, int d, int n) {
    int mx_flow = 0;
    vector<int> parent;
    int flow;
    while(flow = bfs(s, d, n, parent)) {
        mx_flow+=flow;
        int now = d;
        while(now != s) {
            int prev = parent[now];
            capacity[prev][now] -= flow;
            capacity[now][prev] += flow;
            now = prev;
        }
    }
    return mx_flow;
}
bool visited[N+1];
void dfs(int u) {
    visited[u] = true;
    for(auto v : adj[u])if(!visited[v] && capacity[u][v])dfs(v)
      ;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        int n, m;
        cin >> n >> m;
        for(int i = 0; i < m; i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
            capacity[u][v] += 1;
            capacity[v][u] += 1;
        }
        cout << max_flow(1, n, n) << "\n";
        dfs(1);
        for(int u = 1; u <= n; u++) {
            if(visited[u]) {
                for(auto v : adj[u]) {
                    if(!visited[v]) {
                        cout << u << " " << v << "\n";
```

```
                    }
                }
            }
        }
    }
    return 0;
}
```

EdmondsKarp.h
Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

d41d8c, 36 lines

```
template<class T> T edmondsKarp(vector<unordered_map<int, T>>&
    graph, int source, int sink) {
  assert(source != sink);
  T flow = 0;
  vi par(sz(graph)), q = par;

  for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;

    rep(i,0,ptr) {
      int x = q[i];
      for (auto e : graph[x]) {
        if (par[e.first] == -1 && e.second > 0) {
          par[e.first] = x;
          q[ptr++] = e.first;
          if (e.first == sink) goto out;
        }
      }
    }
    return flow;
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
      inc = min(inc, graph[par[y]][y]);

    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
      int p = par[y];
      if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
      graph[y][p] += inc;
    }
  }
}
```

Dinic.h
Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max|cap|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.

d41d8c, 42 lines

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
  };
  vi lvl, ptr, q;
  vector<vector<Edge>> adj;
  Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  void addEdge(int a, int b, ll c, ll rcap = 0) {
    adj[a].push_back({b, sz(adj[b]), c, c});
    adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
  }
  ll dfs(int v, int t, ll f) {
```

```cpp
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < sz(adj[v]); i++) {
      Edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.c))) {
          e.c -= p, adj[e.to][e.rev].c += p;
          return p;
        }
    }
    return 0;
  }
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    rep(L,0,31) do { // 'int L=30' maybe faster for random data
      lvl = ptr = vi(sz(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 - L))
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

MinCut.h
Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

GlobalMinCut.h
Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}\left(V^3\right)$

d41d8c, 21 lines

```cpp
pair<int, vi> globalMinCut(vector<vi> mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i,0,n) co[i] = {i};
  rep(ph,1,n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i,0,n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i,0,n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

GomoryHu.h
Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations
"PushRelabel.h"

d41d8c, 13 lines

```cpp
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i,1,N) {
    PushRelabel D(N); // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j,i+1,N)
      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

7.3 Matching

7.4 DFS algorithms

SCC.h
Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Time: $\mathcal{O}(E+V)$

d41d8c, 29 lines

```cpp
struct SCC {
  // 1-base indexing
  int n;
  vector<vector<int>> adj, radj;
  vector<int> todo, comps, id;
  vector<bool> vis;
  void init(int _n) {
    n = _n;
    adj.resize(n+1), radj.resize(n+1), id.assign(n+1, -1),
      vis.resize(n+1);
  }
  void build(int x, int y) { adj[x].push_back(y), radj[y].
    push_back(x); }
  void dfs(int x) {
    vis[x] = 1;
    for(auto y : adj[x]) if (!vis[y]) dfs(y);
    todo.push_back(x);
  }
  void dfs2(int x, int v) {
    id[x] = v;
    for(auto y : radj[x]) if (id[y] == -1) dfs2(y, v);
  }
  void gen() {
    for(int i = 1; i <= n; i++) if (!vis[i]) dfs(i);
    reverse(todo.begin(), todo.end());
    for(auto x : todo) if (id[x] == -1) {
      dfs2(x, x);
      comps.push_back(x);
    }
  }
} scc;
```

ArticulationPoint.h
Description: Finding articulation points in a graph.

d41d8c, 22 lines

```cpp
vector<int> adj[N];
int t = 0;
vector<int> tin(N, -1), low(N), ap;
void dfs(int u, int p) {
  tin[u] = low[u] = t++;
  int is_ap = 0, child = 0;
  for (int v : adj[u]) {
```

```cpp
    if (v != p) {
      if (tin[v] != -1) {
        low[u] = min(low[u], tin[v]);
      } else {
        child++;
        dfs(v, u);
        if (tin[u] <= low[v]) is_ap = 1;
        low[u] = min(low[u], low[v]);
      }
    }
  }
  if ((p != -1 or child > 1) and is_ap)
    ap.push_back(u);
}
dfs(0, -1);
```

Bridge.h
Description: Finds all the bridges in a graph.

d41d8c, 19 lines

```cpp
void dfs(int v, int p = -1) {
  visited[v] = true;
  tin[v] = low[v] = timer++;
  bool parent_skipped = false;
  for (int to : adj[v]) {
    if (to == p && !parent_skipped) {
      parent_skipped = true;
      continue;
    }
    if (visited[to]) {
      low[v] = min(low[v], tin[to]);
    } else {
      dfs(to, v);
      low[v] = min(low[v], low[to]);
      if (low[to] > tin[v])
        IS_BRIDGE(v, to);
    }
  }
}
```

BiconnectedComponents.h
Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
Usage: int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
Time: $\mathcal{O}(E+V)$

2sat.h
Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ($\sim x$).
Usage: TwoSat ts(number of boolean variables);
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

d41d8c, 80 lines

```cpp
struct _2SAT {
  // 0-base indexing
```

```cpp
int n;
vector<vector<int>> adj, radj;
vector<int> todo, comps, id;
vector<bool> vis, assignment;
void init(int _n) {
    n = _n;
    adj.resize(n), radj.resize(n), id.assign(n, -1), vis.
        resize(n);
    assignment.assign(n/2, false);
}
void build(int x, int y) { adj[x].push_back(y), radj[y].
    push_back(x);}
void dfs1(int x) {
    vis[x] = 1;
    for(auto y : adj[x]) if (!vis[y]) dfs1(y);
    todo.push_back(x);
}
void dfs2(int x, int v) {
    id[x] = v;
    for(auto y : radj[x]) if (id[y] == -1) dfs2(y, v);
}
bool solve_2SAT() {
    for(int i = 0; i < n; i++) if (!vis[i]) dfs1(i);
    reverse(todo.begin(), todo.end());
    int j = 0;
    for(auto x : todo) if (id[x] == -1) {
        dfs2(x, j++);
        // comps.push_back(x);
    }
    for (int i = 0; i < n; i += 2) {
        if (id[i] == id[i + 1]) {
            return false;
        }
        assignment[i / 2] = id[i] > id[i + 1];
    }
    return true;
}
void add_disjunction(int a, bool na, int b, bool nb) {
    // na and nb signify whether a and b are to be negated
    a = 2 * a ^ na;
    b = 2 * b ^ nb;
    int neg_a = a ^ 1;
    int neg_b = b ^ 1;
    build(neg_a, b);
    build(neg_b, a);
}
} _2sat;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        int n, m;
        cin >> n >> m;
        _2sat.init(m*2);
        for(int i = 0; i < n; i++) {
            int a, b;
            char _na, _nb;
            cin >> _na >> a >> _nb >> b;
            bool na, nb;
            --a, --b;
            if(_na == '+')na = false;
            else na = true;
            if(_nb == '+')nb = false;
            else nb = true;
            _2sat.add_disjunction(a, na, b, nb);
```

```cpp
    }
    bool possible = _2sat.solve_2SAT();
    if(possible) {
        for(int i = 0; i < m; i++) {
            if(_2sat.assignment[i])cout <<"+ ";
            else cout << "- ";
        }
    }else cout << "IMPOSSIBLE";
    return 0;
}
}
```

EulerWalk.h
Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V + E)$ d41d8c, 15 lines

```cpp
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        if (it == end){ ret.push_back(x); s.pop_back(); continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.push_back(y);
        }}
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
    return {ret.rbegin(), ret.rend()};
}
```

7.5   Coloring

7.6   Heuristics

7.7   Trees

BinaryLifting.h
Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$ d41d8c, 39 lines

```cpp
const int N = 2e5 + 1;
const int LOG = 18; // LOG = ceil(log2(N))
vector<int> adj[N+1];
int up[N+5][LOG], depth[N+5]; // up[v][j] is the 2^j-th
    Anchestor of node v

void ancestor(int u) {
    for(auto v : adj[u]) {
        depth[v] = depth[u] + 1;
        up[v][0] = u;
        for(int j = 1; j < LOG; j++)up[v][j] = up[up[v][j-1]][j
            -1];
        ancestor(v);
    }
}
int get_lca(int a, int b) {
    if(depth[a] < depth[b])swap(a, b);
    int k = depth[a] - depth[b];
    for(int i = LOG-1; i >= 0; i--)
        if(k & (1 << i))
            a = up[a][i];

    if(a == b)
```

```cpp
        return a;

    for(int i = LOG-1; i >= 0; i--) {
        if(up[a][i] != up[b][i]) {
            a = up[a][i];
            b = up[b][i];
        }
    }
    return up[a][0];
}
/*
int getKthAncestor(int a, int k) {
    for(int i = LOG - 1; i >= 0; i--)
        if(k & (1 << i))
            a = up[a][i];
    return a;
}
*/
```

LCA.h
Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q)$
"../data-structures/RMQ.h"   d41d8c, 20 lines

```cpp
struct LCA {
    int T = 0;
    vi time, path, ret;
    RMQ<int> rmq;

    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
    void dfs(vector<vi>& C, int v, int par) {
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.push_back(v), ret.push_back(time[v]);
            dfs(C, y, v);
        }
    }
    int lca(int a, int b) {
        if (a == b) return a;
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)];
    }
    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};
```

DsuOnTree.h
Description: Dsu on tree
<bits/stdc++.h>   d41d8c, 88 lines

```cpp
using namespace std;
const int N = 2e5 + 1;
int color[N+1];
vector<int> adj[N+1];

int idx = 0, euler[N+1], pos[N+1], sz[N+1], H_C[N+1];

void dfs(int u, int p) {
    pos[u] = idx;
    euler[idx++] = u;
    H_C[u] = -1, sz[u] = 1;
    for(auto v: adj[u]) {
        if(v == p)continue;
        dfs(v, u);
        sz[u]+=sz[v];
        if(H_C[u] == -1 || sz[v] > sz[H_C[u]]) {
            H_C[u] = v;
        }
    }
```

```cpp
}

int freq[N+1], cur_distinct = 0, distinct[N+1];
void add(int u) {
    freq[color[u]]++;
    if(freq[color[u]] == 1)cur_distinct++;
}

void remove(int u) {
    freq[color[u]]--;
    if(freq[color[u]] == 0)cur_distinct--;
}

void dsu(int u, int p, int keep) {
    for(auto v : adj[u]) {
        if(v == p || v == H_C[u]) continue;
        dsu(v, u, 0);
    }
    if(H_C[u] != -1) {
        dsu(H_C[u], u, 1);
    }

    for(auto v : adj[u]) {
        if(v == p || v == H_C[u]) continue;
        for(int i = pos[v]; i < pos[v] + sz[v]; i++) {
            add(euler[i]);
        }
    }
    add(u);
    distinct[u] = cur_distinct;

    if(!keep) {
        for(int i = pos[u]; i < pos[u] + sz[u]; i++) {
            remove(euler[i]);
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        int n;
        cin >> n;
        map<int, int> compress;
        int id = 1;
        for(int i = 1; i <= n; i++) {
            cin >> color[i];
            if(compress[color[i]]) color[i] = compress[color[i
                ]];
            else {
                compress[color[i]] = id++;
                color[i] = compress[color[i]];
            }
        }
        for(int i = 0; i < n-1; i++) {
            int u, v;
            cin >> u >> v;;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        dfs(1, -1);
        dsu(1, -1, 1);
        for(int i = 1; i <= n; i++)cout << distinct[i] << " ";
    }
    return 0;
}
```

```cpp
}
```

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S| - 1$) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.

Time: $\mathcal{O}\left(|S| \log |S|\right)$

"LCA.h"   d41d8c, 21 lines

```cpp
typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
}
```

HLD.h

Description: Heavy Light Decomposition

<bits/stdc++.h>   d41d8c, 136 lines

```cpp
/*
Problem Link: https://cses.fi/problemset/task/2134
*/
using namespace std;
const int N = 2e5 + 1;
int values[N+1], subtree[N+1], parent[N+1], depth[N+1];
int heavy[N+1], head[N+1], id[N+1];
vector<int> adj[N+1];

// 0 Base indexing
struct Segtree {
    int size;
    vector<int> tree;

    int merge(int x, int y) {
        return max(x, y);
    }
    void build(vector<int> &a, int node, int l, int r) {
        if(l == r) {
            tree[node] = a[l];
            return;
        }
        int mid = l + (r - l)/2;
        build(a, node*2+1, l, mid);
        build(a, node*2+2, mid+1, r);
        tree[node] = merge(tree[node*2+1], tree[node*2+2]);
    }
    void update(int i, int value, int node, int l, int r) {
        if(l == i && r == i) {
            tree[node] = value;
            return;
        }
        int mid = l + (r-l)/2;
        if(i <= mid)update(i, value, node*2+1, l, mid);
```

```cpp
        else update(i, value, node*2+2, mid+1, r);
        tree[node] = merge(tree[node*2+1], tree[node*2+2]);
    }
    void update(int i, int value) {
        update(i, value, 0, 0, size-1);
    }
    int query(int i, int j, int node, int l, int r) {
        if(l > j || r < i) return INT_MIN;
        if(l >= i && r <= j)return tree[node];
        int mid = l + (r - l)/2;
        return merge(query(i, j, node*2+1, l, mid), query(i, j,
                node*2+2, mid+1, r));
    }
    int query(int i, int j) {
        return query(i, j, 0, 0, size-1);
    }
    int sz(int n) {
        int size = 1;
        while(size < n) size = size << 1;
        return 2*size-1;
    }
    void init(vector<int> &a, int n) {
        size = 1;
        while(size < n) size = size << 1;
        tree.resize(2*size-1);
        build(a, 0, 0, size-1);
    }
} st;

void dfs(int u, int p) {
    subtree[u] = 1;
    int mx = 0;
    for(auto v : adj[u]) {
        if(v == p)continue;
        parent[v] = u;
        depth[v] = depth[u]+1;
        dfs(v, u);
        subtree[v]+=subtree[u];
        if(subtree[v] > mx) {
            mx = subtree[v];
            heavy[u] = v;
        }
    }
}
int idx = 0;
void HLD(int u, int h) {
    head[u] = h;
    id[u] = idx++;
    if(heavy[u])HLD(heavy[u], h);
    for(auto v : adj[u]) {
        if(v != parent[u] && v != heavy[u]) {
            HLD(v, v);
        }
    }
}
int path(int x, int y) {
    int ans = 0;
    while(head[x] != head[y]) {
        if(depth[head[x]] > depth[head[y]]) swap(x, y);
        ans = max(ans, st.query(id[head[y]], id[y]));
        y = parent[head[y]];
    }
    if(depth[x] > depth[y])swap(x, y);
    ans = max(ans, st.query(id[x], id[y]));
    return ans;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
```

```cpp
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        int n, q;
        cin >> n >> q;
        for(int i = 0; i < n; i++)cin >> values[i];
        for(int i = 0; i < n-1; i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        dfs(1, -1);
        HLD(1, 1);
        vector<int> a(n);
        for(int i = 0; i < n; i++)a[id[i+1]] = values[i];
        st.init(a, n);
        while(q--) {
            int type;
            cin >> type;
            if(type == 1) {
                int s, x;
                cin >> s >> x;
                st.update(id[s], x);
            }else {
                int a, b;
                cin >> a >> b;
                cout << path(a, b) << " ";
            }
        }
    }
    return 0;
}
```

CentroidDecomp.h
Description: Centroid decompose, Finding 1 to K length Path Source :
https://www.codechef.com/problems/PRIMEDST

```cpp
const int N = 50001;
vector<int> adj[N];
int n, k;
int subtree[N], cnt[N], mx_depth, all_cnt[N];
bool visited[N];
// ll ans;

vector<bool> is_prime(N, true);
set<int> primes;
// O(Nlog(log(N)))
void sieve() {
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i * i <= N; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= N; j += i)
                is_prime[j] = false;
        }
    }
}

int getSubtree(int u, int p) {
    subtree[u] = 1;
    for(auto v : adj[u]) {
        if(!visited[v] && v != p) {
            getSubtree(v, u);
            subtree[u]+=subtree[v];
        }
    }
    return subtree[u];
}

int getCentroid(int u, int p, int desired) {
```

```cpp
    for(auto v : adj[u])
        if(!visited[v] && v != p && subtree[v] > desired)
            return getCentroid(v, u, desired);
    return u;
}
void compute(int u, int p, bool filling, int depth) {
    if(depth > k)return;
    mx_depth = max(mx_depth, depth);
    if(filling) {
        cnt[depth]++;
        all_cnt[depth]++;
    }else {
        // ans+=cnt[k - depth]*1LL;
        for(int i = 1; i <= mx_depth; i++) {
            if(cnt[i])all_cnt[i + depth]+=cnt[i];
        }
    }
    for(auto v : adj[u])if(!visited[v] && v != p)compute(v, u,
        filling, depth+1);
}
void centroidDecomposition(int u) {
    int centroid = getCentroid(u, -1, getSubtree(u, -1) >> 1);
    visited[centroid] = true;
    mx_depth = 0;
    for(auto v : adj[centroid]) {
        if(!visited[v]) {
            compute(v, centroid, false, 1);
            compute(v, centroid, true, 1);
        }
    }
    for(int i = 1; i <= mx_depth; i++)cnt[i] = 0;
    for(auto v : adj[centroid])if(!visited[v])
        centroidDecomposition(v);
}
int main() {
    int tt;
    sieve();
    tt = 1;
    // cin >> tt;
    while(tt--) {
        cin >> n;
        for(int i = 2; i <= n-1; i++) {
            if(is_prime[i]) {
                primes.insert(i);
            }
        }
        for(int i = 0; i < n-1; i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        // ans = 0;
        cnt[0] = 1;
        k = *primes.rbegin();
        centroidDecomposition(1);
        ll p_path = 0;
        for(auto x : primes) {
            p_path+=all_cnt[x];
        }
        ll total = n*1LL*(n-1)/2;
        cout << fixed << setprecision(6) << (p_path*1.0)/(total
            *1.0) << "\n";
    }
    return 0;
}
```

LinkCutTree.h
Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$.

```cpp
struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p = p)) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            y->c[h ^ 1] = x;
        }
        z->c[i ^ 1] = this;
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() {
        for (pushFlip(); p; ) {
            if (p->p) p->p->pushFlip();
            p->pushFlip(); pushFlip();
            int c1 = up(), c2 = p->up();
            if (c2 == -1) p->rot(c1, 2);
            else p->p->rot(c2, c1 != c2);
        }
    }
    Node* first() {
        pushFlip();
        return c[0] ? c[0]->first() : (splay(), this);
    }
};

struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v));
        makeRoot(&node[u]);
        node[u].pp = &node[v];
    }
    void cut(int u, int v) { // remove an edge (u, v)
        Node *x = &node[u], *top = &node[v];
        makeRoot(top); x->splay();
        assert(top == (x->pp ?: x->c[0]));
        if (x->pp) x->pp = 0;
        else {
            x->c[0] = top->p = 0;
            x->fix();
        }
    }
```

```
    }
  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

Time: $\mathcal{O}(E \log V)$

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
  ll res = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
```

```
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
  }
  for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  rep(i,0,n) par[i] = in[i].a;
  return {res, par};
}
```

DPOnTree.h

Description: DPonTree

```
const int N = 100000;
int n, mod;
vector<int> adj[N];
// up[i] = total ways to paint all the ancestors of node i
// if the parent of node i is painted black.
vector<ll> up(N, 1);
// down[i] = total ways to paint the subtree of node i
// if the node i is painted black or white.
ll down[N];

void dfs1(int u, int parent) {
  down[u] = 1;
  for(auto v : adj[u]) {
    if(v == parent)continue;
    dfs1(v, u);
    down[u] = (down[u] * down[v]) % mod;
  }
  down[u] = (down[u] + 1) % mod;
}

void dfs2(int u, int parent) {
  int pref = 1;
  for(auto v : adj[u]) {
    if(v == parent)continue;
    up[v] = pref % mod;
    pref = pref*down[v] % mod;
  }
  reverse(adj[u].begin(), adj[u].end());
  int suff = 1;
  for(auto v : adj[u]) {
    if(v == parent)continue;
    up[v] = up[v]*suff % mod;
    suff = suff*down[v] % mod;
  }
  for(auto v : adj[u]) {
    if(v == parent)continue;
    up[v] = up[u] * up[v] % mod;
    up[v] = (up[v] + 1) % mod;
    dfs2(v, u);
  }
}
int main() {
  ios::sync_with_stdio(false);
```

```
  cin.tie(0);
  int tt;
  tt = 1;
  // cin >> tt;
  while(tt--) {
    cin >> n >> mod;
    for(int i = 0; i < n-1; i++) {
      int u, v;
      cin >> u >> v;
      --v, --u;
      adj[u].push_back(v);
      adj[v].push_back(u);
    }
    dfs1(0, -1);
    dfs2(0, -1);
    for(int i = 0; i < n; i++) {
      cout << up[i]*(down[i] - 1 + mod) % mod << "\n";
    }
  }
  return 0;
}
```

7.8 Math

7.8.1 Number of Spanning Trees

Create an $N \times N$ matrix mat, and for each edge $a \to b \in G$, do mat[a][b]--, mat[b][b]++ (and mat[b][a]--, mat[a][a]++ if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

7.8.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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

Geometry (8)

8.1 Geometric primitives

Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

```
using ftype = ll;
const double eps = 1e-9;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps);}

struct P {
  ftype x, y;
  P() {}
  P(ftype x, ftype y): x(x), y(y) {}
  void read() {
    cin >> x >> y;
  }
  P& operator+=(const P &t) {
    x += t.x;
    y += t.y;
    return *this;
```

```cpp
        }
        P& operator-=(const P &t) {
            x -= t.x;
            y -= t.y;
            return *this;
        }
        P& operator*=(ftype t) {
            x *= t;
            y *= t;
            return *this;
        }
        P& operator/=(ftype t) {
            x /= t;
            y /= t;
            return *this;
        }
        P operator+(const P &t) const {return P(*this) += t;}
        P operator-(const P &t) const {return P(*this) -= t;}
        P operator*(ftype t) const {return P(*this) *= t;}
        P operator/(ftype t) const {return P(*this) /= t;}
        bool operator == (P a) const { return sign(a.x - x) == 0 &&
            sign(a.y - y) == 0; }
        bool operator != (P a) const { return !(*this == a); }
        bool operator < (P a) const { return sign(a.x - x) == 0 ? y
            < a.y : x < a.x; }
        bool operator > (P a) const { return sign(a.x - x) == 0 ? y
            > a.y : x > a.x; }
        P perp() const {
            return P(y, -x); // Or P(y, -x) depending on the
                desired direction.
        }
};

P operator*(ftype a, P b) {return b * a;}
inline ftype dot(P a, P b) {return a.x * b.x + a.y * b.y;}
inline ftype cross(P a, P b) {return a.x * b.y - a.y * b.x;}
ftype norm(P a) {return dot(a, a);}
double abs(P a) {return sqrt(norm(a));}
double proj(P a, P b) {return dot(a, b) / abs(b);}
double angle(P a, P b) {return acos(dot(a, b) / abs(a) / abs(b)
    );}
P intersect(P a1, P d1, P a2, P d2) {return a1 + cross(a2 - a1,
    d2) / cross(d1, d2) * d1;}

bool LineSegmentIntersection(P p1, P p2, P p3, P p4) {
    // Check if they are parallel
    if(cross(p1-p2, p3-p4) == 0) {
        // If they are not collinear
        if(cross(p2-p1, p3-p1) != 0) {
            return false;
        }
        // Check if they are collinear and do not intersect
        for(int it = 0; it < 2; it++) {
            if(max(p1.x, p2.x) < min(p3.x, p4.x) ||
                max(p1.y, p2.y) < min(p3.y, p4.y)) {
                return false;
            }
            swap(p1, p3), swap(p2, p4);
        }
        return true;
    }
    // Check one segment totally on the left or right side of
        other segment
    for(int it = 0; it < 2; it++) {
        ll sign1 = cross(p2-p1, p3-p1);
        ll sign2 = cross(p2-p1, p4-p1);
        if((sign1 < 0 && sign2 < 0) || (sign1 > 0 && sign2 > 0)
            ) {
            return false;
        }
```

```cpp
        }
        swap(p1, p3), swap(p2, p4);
    }
    // For all other case return true
    return true;
}

// here return value is area*2
ftype PolygonArea(vector<P> &polygon, int n) {
    ll area = 0;
    for(int i = 0; i < n; i++) {
        int j = (i+1) % n;
        area+=cross(polygon[i], polygon[j]);
    }
    return abs(area);
}

string PointInPolygon(vector<P> &polygon, int n, P &p) {
    int cnt = 0;
    for(int i = 0; i < n; i++) {
        int j = (i+1) % n;
        if(LineSegmentIntersection(polygon[i], polygon[j], p, p
            )) {
            return "BOUNDARY";
        }
        /*
        Imagine a vertically infinite line from point p to
            positive infinity.
        Check if a line from the polygon is totally on the left
            or right side of the infinite line and makes a
            positive cross product or positive triangle.
        Here, "right" means to the right or equal.
        */
        if((polygon[i].x >= p.x && polygon[j].x < p.x && cross(
            polygon[i]-p, polygon[j]-p) > 0) ||
            (polygon[i].x < p.x && polygon[j].x >= p.x && cross(
            polygon[j]-p, polygon[i]-p) > 0))
            cnt++;
    }
    if(cnt & 1)return "INSIDE";
    return "OUTSIDE";
}


void ConvexHull(vector<P> &points, int n) {
    vector<P> hull;
    sort(points.begin(), points.end());
    for(int rep = 0; rep < 2; rep++) {
        const int h = (int)hull.size();
        for(auto C : points) {
            while((int)hull.size() - h >= 2) {
                P A = hull[(int)hull.size()-2];
                P B = hull[(int)hull.size()-1];
                if(cross(B-A, C-A) <= 0) {
                    break;
                }
                hull.pop_back();
            }
            hull.push_back(C);
        }
        hull.pop_back();
        reverse(points.begin(), points.end());
    }
    cout << hull.size() << "\n";
    for(auto p : hull) {
        cout << p.x << " " << p.y << "\n";
    }
}
bool circleInter(P a, P b, double r1, double r2, pair<P, P>*
    out) {
```

```cpp
    P vec = b - a;
    double d2 = norm(vec);
    double d = sqrt(d2);
    if (d > r1 + r2 || d < fabs(r1 - r2)) {
        return false;
    }
    double p = (d2 + r1 * r1 - r2 * r2) / (2 * d);
    double h2 = r1 * r1 - p * p;
    if (h2 < 0) h2 = 0;
    P mid = a + vec * (p / d);
    P per = vec.perp() * (sqrt(h2) / d);
    *out = {mid + per, mid - per};
    return true;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        int n;
        cin >> n;
        vector<P> points;
        for(int i = 0; i < n; i++) {
            P p;
            p.read();
            points.push_back(p);
        }
        ConvexHull(points, n);
    }
    return 0;
}
```

8.2   Circles

8.3   Misc. Point Set Problems

ClosestPair.h
Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$
<div align="right">d41d8c, 64 lines</div>

```cpp
#define pii pair<ll, ll>
#define ff first
#define ss second

bool comparex(pii a, pii b) { return a.first < b.first; }
bool comparey(pii a, pii b) { return a.second < b.second; }
ll dist(pii x, pii y) { return (x.ff - y.ff) * (x.ff - y.ff) +
    (x.ss - y.ss) * (x.ss - y.ss); }
pair<pii, pii> closestAmongThree(pii a, pii b, pii c) {
    ll d1 = dist(a, b);
    ll d2 = dist(b, c);
    ll d3 = dist(a, c);
    ll mn = min({ d1, d2, d3 });
    if (mn == d1) return { a, b };
    else if (mn == d2) return { b, c };
    else return { a, c };
}
pair<pii, pii> closest(vector<pii>& points, ll st, ll en) {
    if (st + 1 == en) return { points[st], points[en] };
    if (st + 2 == en) return closestAmongThree(points[st],
        points[st + 1], points[en]);

    ll mid = st + (en - st) / 2;

    pair<pii, pii> left = closest(points, st, mid);
    pair<pii, pii> right = closest(points, mid + 1, en);
    ll left_d = dist(left.ff, left.ss);
    ll right_d = dist(right.ff, right.ss);
```

```
    ll d = min(left_d, right_d);
    pair<pii, pii> ans = (d == left_d) ? left : right;

    vector<pii> middle;
    for (int i = st; i <= en; i++)
        if (abs(points[i].ff - points[mid].ff) < d)
            middle.push_back(points[i]);
    sort(middle.begin(), middle.end(), comparey);

    for (int i = 0; i < (int)middle.size(); i++) {
        for (int j = i + 1; j < (int)middle.size() and (middle[
            j].ss - middle[i].ss) * (middle[j].ss - middle[i].
            ss) < d; j++) {
            ll dst = dist(middle[i], middle[j]);
            if (dst < d) {
                ans = { middle[i], middle[j] };
                d = dst;
            }
        }
    }
    middle.clear();

    return ans;
}
int main() {
    int tt;
    tt = 1;
    while (tt--) {
        int n;
        cin >> n;
        vector<pii> points(n);
        for (int i = 0; i < n; i++) {
            cin >> points[i].first >> points[i].second;
        }
        sort(points.begin(), points.end(), comparex);
        pair<pii, pii> ans = closest(points, 0, n - 1);
        cout << dist(ans.ff, ans.ss) << '\n';
    }
    return 0;
}
```

SweepLine.h
Description: Returns any intersecting segments, or -1, -1 if none exist.
Time: $\mathcal{O}(N \log N)$

Strings (9)

KMP.h
Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.
Time: $\mathcal{O}(n)$

<div align="right">d41d8c, 15 lines</div>

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}
vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

Zfunc.h
Description: z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

<div align="right">d41d8c, 12 lines</div>

```
vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

Manacher.h
Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$

<div align="right">d41d8c, 13 lines</div>

```
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi,2> p = {vi(n+1), vi(n)};
  rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
    int t = r-i+!z;
    if (i<r) p[z][i] = min(t, p[z][l+t]);
    int L = i-p[z][i], R = i+p[z][i]-!z;
    while (L>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
  }
  return p;
}
```

MinRotation.h
Description: Finds the lexicographically smallest rotation of a string.
Usage: rotate(v.begin(), v.begin()+minRotation(v), v.end());
Time: $\mathcal{O}(N)$

<div align="right">d41d8c, 8 lines</div>

```
int minRotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(k,0,N) {
    if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
    if (s[a+k] > s[b+k]) { a = b; break; }
  }
  return a;
}
```

SuffixArray.h
Description: Suffix Array

<div align="right">d41d8c, 44 lines</div>

```
void count_sort(vector<pli> &b, int bits) {
  int mask = (1 << bits) - 1;
  rep(it, 0, 2) {
    int shift = it * bits;
    vi q(1 << bits), w(sz(q) + 1);
    rep(i, 0, sz(b)) q[(b[i].first >> shift) & mask]++;
    partial_sum(q.begin(), q.end(), w.begin() + 1);
    vector<pli> res(sz(b));
    rep(i, 0, sz(b)) res[w[(b[i].first >> shift) & mask]++] = b
      [i];
    swap(b, res);
  }
}
```

```
}
struct SuffixArray {
  vi a; string s;
  SuffixArray(const string &str) : s(str + '\0') {
    int N = sz(s), q = 8;
    while ((1 << q) < N) q++;
    vector<pli> b(N);
    a.resize(N);
    rep(i, 0, N) b[i] = {s[i], i};
    for (int moc = 0;; moc++) {
      count_sort(b, q);
      rep(i, 0, N) a[b[i].second] = (i && b[i].first == b[i -
        1].first) ? a[b[i - 1].second] : i;
      if ((1 << moc) >= N) break;
      rep(i, 0, N) {
        b[i] = {(ll)a[i] << q, i + (1 << moc) < N ? a[i + (1 <<
          moc)] : 0};
        b[i].second = i;
      }
    }
    rep(i, 0, N) a[i] = b[i].second;
  }
  vi lcp() {
    int n = sz(a), h = 0;
    vi inv(n), res(n);
    rep(i, 0, n) inv[a[i]] = i;
    rep(i, 0, n) if (inv[i]) {
      int p0 = a[inv[i] - 1];
      while (s[i + h] == s[p0 + h]) h++;
      res[inv[i]] = h;
      if (h) h--;
    }
    return res;
  }
};
```

SuffixTree.h
Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$

<div align="right">d41d8c, 47 lines</div>

```
struct SuffixTree {
  enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
  int toi(char c) { return c - 'a'; }
  string a; // v = cur node, q = cur position
  int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;
  void ukkadd(int i, int c) { suff:
    if (r[v]<=q) {
      if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
        p[m++]=v; v=s[v]; q=r[v]; goto suff; }
      v=t[v][c]; q=l[v];
    }
    if (q==-1 || c==toi(a[q])) q++; else {
      l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
      p[m]=p[v]; t[m][toi(a[q])]=v;
      l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
      v=s[p[m]]; q=l[m];
      while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }
      if (q==r[m]) s[m]=v; else s[m]=m+2;
      q=r[v]-(q-r[m]); m+=2; goto suff;
    }
  }
  SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
```

```
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
  }
  // example: find longest common substring (uses ALPHA = 28)
  pii best;
  int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
      mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
      best = max(best, {len, r[node] - len});
    return mask;
  }
  static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
  }
};
```

Hashing.h
Description: Self-explanatory methods for string hashing.(Arithmetic mod $2^{64} - 1$. 2x slower than mod 2^{64} and more code, but works on evil test data (e.g. Thue-Morse, where ABBA... and BAAB... of length 2^{10} hash the same mod 2^{64}). "typedef ull H;" instead if you think test data is random, or work mod $10^9 + 7$ if the Birthday paradox is not a problem.)
    d41d8c, 36 lines

```
typedef uint64_t ull;
struct H {
  ull x; H(ull x=0) : x(x) {}
  H operator+(H o) { return x + o.x + (x + o.x < x); }
  H operator-(H o) { return *this + ~o.x; }
  H operator*(H o) { auto m = (__uint128_t)x * o.x;
    return H((ull)m) + (ull)(m >> 64); }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get() == o.get(); }
  bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)
struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};
vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}
H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

AhoCorasick.h
Description: Aho Corasick
    d41d8c, 56 lines

```
struct AC {
  int N, P;
  const int A = 26;
  vector <vector <int>> next;
  vector <int> link, out_link;
  vector <vector <int>> out;
  AC(): N(0), P(0) {node();}
  int node() {
    next.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }
  inline int get (char c) {
    return c - 'a';
  }
  int add_pattern (const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][get(c)]) next[u][get(c)] = node();
      u = next[u][get(c)];
    }
    out[u].push_back(P);
    return P++;
  }
  void compute() {
    queue <int> q;
    for (q.push(0); !q.empty();) {
      int u = q.front(); q.pop();
      for (int c = 0; c < A; ++c) {
        int v = next[u][c];
        if (!v) next[u][c] = next[link[u]][c];
        else {
          link[v] = u ? next[link[u]][c] : 0;
          out_link[v] = out[link[v]].empty() ? out_link[link[v
            ]] : link[v];
          q.push(v);
        }
      }
    }
  }
  int advance (int u, char c) {
    while (u && !next[u][get(c)]) u = link[u];
    u = next[u][get(c)];
    return u;
  }
  void match (const string S) {
    int u = 0;
    for (auto c : S) {
      u = advance(u, c);
      for (int v = u; v; v = out_link[v]) {
        for (auto p : out[v]) cout << "match " << p << endl;
      }
    }
  }
};
```

Various (10)

10.1   Intervals

IntervalContainer.h
Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$
    d41d8c, 23 lines

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->second;
  if (it->first == L) is.erase(it);
  else (int&)it->second = L;
  if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h
Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive]. To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$
    d41d8c, 19 lines

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
  vi S(sz(I)), R;
  iota(all(S), 0);
  sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
  T cur = G.first;
  int at = 0;
  while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <= cur) {
      mx = max(mx, make_pair(I[S[at]].second, S[at]));
      at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

ConstantIntervals.h
Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$
    d41d8c, 19 lines

```
template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
  if (p == q) return;
  if (from == to) {
    g(i, to, p);
```

```
    i = to; p = q;
  } else {
    int mid = (from + to) >> 1;
    rec(from, mid, f, g, i, p, f(mid));
    rec(mid+1, to, f, g, i, p, q);
  }
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
  if (to <= from) return;
  int i = from; auto p = f(i), q = f(to-1);
  rec(from, to-1, f, g, i, p, q);
  g(i, to, q);
}
```

10.2   Misc. algorithms

TernarySearch.h
Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b - a))$
<div align="right">d41d8c, 11 lines</div>

```
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) a = mid; // (A)
    else b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

LIS.h
Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$
<div align="right">d41d8c, 17 lines</div>

```
template<class I> vi lis(const vector<I>& S) {
  if (S.empty()) return {};
  vi prev(sz(S));
  typedef pair<I, int> p;
  vector<p> res;
  rep(i,0,sz(S)) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0});
    if (it == res.end()) res.emplace_back(), it = res.end()-1;
    *it = {S[i], i};
    prev[i] = it == res.begin() ? 0 : (it-1)->second;
  }
  int L = sz(res), cur = res.back().second;
  vi ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  return ans;
}
```

FastKnapsack.h
Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$
<div align="right">d41d8c, 16 lines</div>

```
int knapsack(vi w, int t) {
  int a = 0, b = 0, x;
  while (b < sz(w) && a + w[b] <= t) a += w[b++];
  if (b == sz(w)) return a;
  int m = *max_element(all(w));
```

```
  vi u, v(2*m, -1);
  v[a+m-t] = b;
  rep(i,b,sz(w)) {
    u = v;
    rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
    for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
      v[x-w[j]] = max(v[x-w[j]], j);
  }
  for (a = t; v[a+m-t] < 0; a--) ;
  return a;
}
```

10.3   Dynamic programming

KnuthDP.h
Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i, j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

DivideAndConquerDP.h
Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)}(f(i, k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R - 1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$
<div align="right">d41d8c, 18 lines</div>

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

10.4   Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). _GLIBCXX_DEBUG failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).

- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

10.5   Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

10.5.1   Bit hacks

- `x & -x` is the least bit in `x`.

- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).

- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.

- `rep(b,0,K) rep(i,0,(1 << K))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];`
  computes all sums of subsets.

10.5.2   Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.

- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.

- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

FastMod.h
Description: Compute $a\%b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod b$ in the range $[0, 2b)$.
<div align="right">d41d8c, 8 lines</div>

```
typedef unsigned long long ull;
struct FastMod {
  ull b, m;
  FastMod(ull b) : b(b), m(-1ULL / b) {}
  ull reduce(ull a) { // a % b + (0 or b)
    return a - (ull)((__uint128_t(m) * a) >> 64) * b;
  }
};
```

10.6   Miscellaneous

SOSDP.h
Description: SOS DP
<div align="right">d41d8c, 10 lines</div>

```
vector<vector<int>> dp(1 << n, vector<int>(n));
vector<int> sos(1 << n);
for (int mask = 0; mask < (1 << n); mask++) {
  dp[mask][-1] = a[mask];
  for (int x = 0; x < n; x++) {
    dp[mask][x] = dp[mask][x - 1];
    if (mask & (1 << x)) { dp[mask][x] += dp[mask - (1 << x)][x
      - 1]; }
  }
  sos[mask] = dp[mask][n - 1];
}
```

submaskiterate.h
Description: Submask iterate
<div align="right">d41d8c, 3 lines</div>

```
for (int m=0; m<(1<<n); ++m)
  for (int s=m; s; s=(s-1)&m)
  ... s and m ...
```

nCrNotP.h
Description: Finds nCr modulo a number that is not necessarily prime. Its good when m is small and not fixed.

Time: $\mathcal{O}(m \log m)$

"../number-theory/CRT.h", "../number-theory/ModPow.h"   d41d8c, 32 lines

```cpp
int F[1000002] = {1}, p, e, pe;
ll lg(ll n, int p) {
  ll r = 0;
  while (n) n /= p, r += n;
  return r;
}
ll f(ll n) {
  if (!n) return 1;
  return modpow(F[pe], n / pe, pe) * (F[n % pe] * f(n / p) % pe
      ) % pe;
}
ll ncr(ll n, ll r) {
  ll c;
  if ((c = lg(n, p) - lg(r, p) - lg(n - r, p)) >= e)
    return 0;
  for (int i = 1; i <= pe; i++)
    F[i] = F[i - 1] * (i % p == 0 ? 1 : i) % pe;
  return (f(n) * modpow(p, c, pe) % pe) *
    modpow(f(r) * f(n - r), pe - (pe / p) - 1, pe) % pe;
}
ll ncr(ll n, ll r, ll m) {
  ll a0 = 0, m0 = 1;
  for (p = 2; m != 1; p++) {
    e = 0, pe = 1;
    while (m % p == 0)
      m /= p, e++, pe *= p;
    if (e) {
      a0 = crt(a0, m0, ncr(n, r), pe);
      m0 = m0 * pe;
    }
  }
  return a0;
}
```