University of Dhaka

DU_NE

Emon Khan, Himel Roy, Syed Waki As Sami

2025-01-17

Contest (1)

template.cpp
17 lines

```
#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

```
#!/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

```
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

```
# 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 $r = \sqrt{a^2+b^2}, \phi = \operatorname{atan2}(b, a)$.

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: $\dfrac{a+b}{a-b} = \dfrac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$

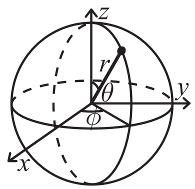
2.4.2 Quadrilaterals

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



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

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx} \tan x = 1 + \tan^2 x \qquad \frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$
$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x) \qquad \int x e^{ax} dx = \frac{e^{ax}}{a^2}(ax - 1)$$

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 $\mathrm{U}(a, b)$, $a < b$.

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

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

Exponential distribution

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

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

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

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

```
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

```
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

```
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

```
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;
```

PersistentSegtree.h

Description: PresistentSegment Tree

d41d8c, 76 lines

```
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) {
        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;
}
```

UnionFind.h

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

d41d8c, 17 lines

```
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: $\mathcal{O}\left(N^2 + Q\right)$

d41d8c, 34 lines

```
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

```
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

```
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();
        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

```
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

```
struct FenwickTree2D {
    // 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

```
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 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$

d41d8c, 15 lines

```
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;
}
```

Number theory (5)

5.1 Modular arithmetic

ModPow.h

d41d8c, 11 lines

```
int bigPow(ll base, ll power, const int mod) {
    int ans = 1 % mod;
    base %= mod;
    if (base < 0) base += mod;
    while (power) {
        if (power & 1) ans = (ll) ans * base % mod;
        base = (ll) base * base % mod;
        power >>= 1;
    }
    return ans;
}
```

MatrixExpo.h

<bits/stdc++.h>

d41d8c, 52 lines

```
// https://codeforces.com/gym/102644/problem/C
using namespace std;
#define ll long long
const int M = 1e9 + 7;

struct Matrix {
    int a[2][2] = {{0, 0}, {0, 0}};
    Matrix operator *(const Matrix& other) {
        Matrix product;
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                for (int k = 0; k < 2; k++) {
                    product.a[i][k] = (product.a[i][k] + (ll) a
                        [i][j] * other.a[j][k]) % M;
                }
            }
        }
        return product;
    }
};

Matrix expo_power(Matrix a, ll k) {
    Matrix product;
    for (int i = 0; i < 2; i++) {
        product.a[i][i] = 1;
    }
    while (k > 0) {
        if (k % 2) {
            product = product * a;
        }
        a = a * a;
        k /= 2;
    }
    return product;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        ll k;
        cin >> k;
        Matrix M;
        M.a[0][0] = 1;
        M.a[0][1] = 1;
        M.a[1][0] = 1;
        M.a[1][1] = 0;
        cout << expo_power(M, k).a[1][0] << "\n";
    }
    return 0;
}
```

SumProductCountOfDivisors.h

<bits/stdc++.h>

d41d8c, 57 lines

```
/*
Problem Link: https://cses.fi/problemset/task/2182/
*/
using namespace std;
const int M = 1e9 + 7;
#define ll long long

int bigPow(ll base, ll power, const int mod) {
    int ans = 1 % mod;
    base %= mod;
    if (base < 0) base += mod;
    while (power) {
        if (power & 1) ans = (ll) ans * base % mod;
        base = (ll) base * base % mod;
        power >>= 1;
    }
    return ans;
}
// S_n = a(1-r^n)/(1-r)
int geometricSeriesSum(int r, int n) {
    int nu = bigPow(r, n, M) - 1; // Numerator
    int de = r - 1; // Denominator
    de = bigPow(de, M-2, M);
    return nu*1LL*de % M;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        int n;
        cin >> n;
        ll cnt = 1, sum = 1, prod = 1, num1 = 1, num2 = 1, pw =
            1;
        bool ok = true;
        for(int i = 0; i < n; i++) {
            int x, k;
            cin >> x >> k;
            cnt = cnt * (k + 1) % M;
            sum = sum * geometricSeriesSum(x, k+1) % M;
            num1 = num1 * bigPow(x, k, M) % M;
            num2 = num2 * bigPow(x, k/2, M) % M;
            if(k % 2 != 0 && ok) {
                pw = (pw * (k+1)/2) % (M-1);
                ok = false;
            }else {
```

```
            pw = (pw * (k+1)) % (M-1);
        }
    }
    // Product of divisors = (Num)^(d(Num)/2)
    if(!ok)prod = bigPow(num1, pw, M);
    else prod = bigPow(num2, pw, M);
    cout << cnt << " " << sum << " " << prod << "\n";
  }
  return 0;
}
```

Sieve.h

<bits/stdc++.h> d41d8c, 105 lines

```
using namespace std;
#define ll long long
const int N = 100000;
vector<bool> is_prime(N+1, true);

// O(Nlog(N))
void divisors() {
    vector<vector<int>> d(N+1);
    for(int i = 1; i <= N; i++) {
        for(int j = i; j <= N; j+=i) {
            d[j].push_back(i);
        }
    }
}
// O(sqrt(N))
vector<ll> divisor(ll a) {
    vector<ll> divisors;
    for (ll i = 1; i*i <= a; ++i) {
        if(a % i == 0) {
            if(a / i == i)divisors.push_back(i);
            else {
                divisors.push_back(i);
                divisors.push_back(a/i);
            }
        }
    }
    return divisors;
}
// 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;
        }
    }
}
// O(sqrt(N))
vector<ll> prime_factorization(ll n) {
    vector<ll> factorization;
    while (n % 2 == 0) {
        factorization.push_back(2);
        n /= 2;
    }
    for (ll d = 3; d * d <= n; d += 2) {
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
    }
    if (n > 1) factorization.push_back(n);
    return factorization;
}
// O(sqrt(N))
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;
}
// O(Nloglog(N))
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
// O(Nloglog(N))
void phi_1_to_n_(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
              phi[j] -= phi[i];
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    tt = 1;
    // cin >> tt;
    while(tt--) {
        int n;
        cin >> n;
    }
    return 0;
}
```

5.2 Primality

5.3 Divisibility

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

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

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

5.3.2 Bézout's identity

For $a \neq, b \neq 0$, 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

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

5.4 Fractions

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 Fibonacci

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:

$$\begin{aligned} F_{n-1}F_{n+1} - F_n^2 &= (-1)^n & F_{n+k} &= F_{k-1}F_n + F_kF_{n+1} \\ F_{2n} &= F_n(F_{n-1} + F_{n+1}) & F_{2n+1} &= F_n^2 + F_{n+1}^2 \\ n|m &\Leftrightarrow F_n|F_m & \gcd(F_m, F_n) &= F_{\gcd(m,n)} \end{aligned}$$

Fibonacci.h

Description: nthFibonacci

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

d41d8c, 8 lines

```
ll f(ll n) {
    if(n == 0 || n == 1)return dp[n] = 1;
    if(dp[n])return dp[n];
    ll k = n/2;
    if(n % 2 == 0)return dp[n] = (f(k)*f(k) + f(k-1)*f(k-1)) %
        M;
    return dp[n] = (f(k)*f(k+1) + f(k-1) * f(k)) % M;
}
(n == 0 ? 0 : f(n-1));
```

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

```
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

```
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.

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

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

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. $\pi(j) \ge j$, 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
    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 $m[i][j] =$ `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]);
                }
            }
        }
    }
}
```

Dijkstra.h

Description: Dijstra d41d8c, 22 lines

```cpp
vector<ll> dijkstra(int s, int n, vector<vector<pair<int, ll>>>
     &adj) {
    vector<ll> dist(n+1, INF);
    dist[s] = 0;
    priority_queue<pair<ll, int>, vector<pair<ll, int>>,
         greater<pair<ll, int>>> pq;
    pq.push({0, s});
    bool vis[n+1];
    memset(vis, false, sizeof(vis));
    while(!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();
        if(vis[u])continue;
        vis[u] = true;
        for(auto [v, wt] : adj[u]) {
            ll _d = d + wt;
            if(_d < dist[v]) {
                dist[v] = _d;
                pq.push({_d, v});
            }
        }
    }
    return dist;
}
```

7.2 Network flow

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: $\mathcal{O}(FE\log(V))$ where F is max flow. $\mathcal{O}(VE)$ for setpi. d41d8c, 76 lines

```cpp
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;
}
```

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

```
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

```
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]) {
    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

```
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);
    }
  }
}
```

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 $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (∼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

```
struct _2SAT {
    // 0-base indexing
    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);
        }
        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: $\mathcal{O}(V+E)$

<bits/stdc++.h>, <bits/stdc++.h> d41d8c, 114 lines

```
/*
Problem Link: https://cses.fi/problemset/task/1691/
Idea: Euler Circuit in undirected graph Hierholzer Algorithm
*/
using namespace std;
const int N = 100000;
vector<pair<int, int>> adj[N+1];
int degree[N+1];
bool visited[2*N+1]; // total edge size
vector<int> euler_path;

void dfs(int u) {
    while(!adj[u].empty()) {
        auto [v, idx] = adj[u].back();
        adj[u].pop_back();
        if(visited[idx])continue;
        visited[idx] = true;
        dfs(v);
    }
    euler_path.push_back(u);
}
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, i});
            adj[v].push_back({u, i});
            degree[u]++, degree[v]++;
        }
        /*
        Undirected Graphs:
        Euler Circuit: All vertices must have even degree.
        Euler Path: Exactly zero or two vertices can have odd
            degree.
        */
        for(int i = 1; i <= n; i++) {
            if(degree[i] % 2 != 0) {
                cout << "IMPOSSIBLE\n";
                return 0;
            }
        }
        dfs(1);
        if(euler_path.size() !=  m+1) {
            cout << "IMPOSSIBLE\n";
            return 0;
        }
        for(auto x : euler_path) {cout << x << " ";}
    }
    return 0;
}
/*
Problem Link: https://cses.fi/problemset/task/1693/
Idea: Euler Path in Directed graph Hierholzer Algorithm
*/
using namespace std;
const int N = 100000;
vector<int> adj[N+1];
int in[N+1], out[N+1];
vector<int> euler_path;

void dfs(int u) {
    while(!adj[u].empty()) {
        int v = adj[u].back();
        adj[u].pop_back();
        dfs(v);
    }
    euler_path.push_back(u);
}
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);
            out[u]++, in[v]++;
        }
        /*
        Directed Graphs:
        Euler Circuit: All vertices must have equal in-degree
            and out-degree.
        Euler Path: Exactly two vertices can have a difference
            of one between their in-degree and out-degree.
        */
        for(int i = 1; i <= n; i++) {
            if((i == 1 && out[1]-in[1] != 1) ||
                (i == n && in[n]-out[n] != 1) ||
                (i > 1 && i < n && out[i] != in[i])) {
                cout << "IMPOSSIBLE\n";
                return 0;
            }
        }
        dfs(1);
        reverse(euler_path.begin(), euler_path.end());
        if(euler_path.size() - 1 !=  m || euler_path.back() !=
            n) {
            cout << "IMPOSSIBLE\n";
            return 0;
        }
        for(auto x : euler_path) {cout << x << " ";}
    }
    return 0;
}
```

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

```
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)
        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

```
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

```
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;
        }
    }
}

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;
}
```

HLD.h

Description: Heavy Light Decomposition

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

```
/*
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);
        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);
    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

d41d8c, 93 lines

```
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) {
    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;
}
```

DPOnTree.h

Description: DPonTree

d41d8c, 64 lines

```
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.)

d41d8c, 172 lines

```
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;
    }
    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;
        }
        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) {
    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: $\mathcal{O}(n \log n)$

d41d8c, 64 lines

```
#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: $\mathcal{O}(n)$

d41d8c, 13 lines

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

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: $\mathcal{O}(n)$

d41d8c, 18 lines

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + 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: $\mathcal{O}(N)$

d41d8c, 31 lines

```
vector<int> manacher(string t) {
    string s;
    for(auto c: t) {
        s += string("#") + c;
    }
    s+="#";
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}

// 0-base indexing
bool is_palindrome(int l, int r, vector<int> &pal) {
    l++, r++;
    int range = (r - l) + 1;
    l = (l << 1) - 1;
    r = (r << 1) - 1;
    int mid = (l + r) >> 1;
    return pal[mid] >= range;
}
```

MinRotation.h

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

d41d8c, 24 lines

```
int least_rotation(const string &s) {
    int n = s.length();
    vector<int> f(2 * n, -1);
    int k = 0;

    for (int j = 1; j < 2 * n; ++j) {
        int i = f[j - k - 1];
        while (i != -1 && s[j % n] != s[(k + i + 1) % n]) {
            if (s[j % n] < s[(k + i + 1) % n]) {
                k = j - i - 1;
            }
            i = f[i];
```

```
        }
        if (i == -1 && s[j % n] != s[(k + i + 1) % n]) {
            if (s[j % n] < s[(k + i + 1) % n]) {
                k = j;
            }
            f[j - k] = -1;
        } else {
            f[j - k] = i + 1;
        }
    }
    return k;
}
```

SuffixArray.h

Description: Suffix Array

<bits/stdc++.h> d41d8c, 109 lines

```
/*
Problem Name: Finding Patterns
Problem Link: https://cses.fi/problemset/task/2102/
Idea: Suffix Array
Complexity:
Resource:
*/
using namespace std;

void radix_sort(vector<int> &p, vector<int> &c) {
  int n = p.size();

  vector<int> cnt(n);
  for (auto x : c) {
    cnt[x]++;
  }

  vector<int> p_new(n);
  vector<int> pos(n);
  pos[0] = 0;
  for (int i = 1; i < n; i++) {
    pos[i] = pos[i - 1] + cnt[i - 1];
  }

  for (auto x : p) {
    int i = c[x];
    p_new[pos[i]] = x;
    pos[i]++;
  }
  p = p_new;
}
void SA() {
  string s;
  cin >> s;
  s += "$";
  int n = s.size();
  vector<int> p(n), c(n);

    // k = 0
  vector<pair<char, int>> a(n);
  for (int i = 0; i < n; i++) a[i] = {s[i], i};
    sort(a.begin(), a.end());
  for (int i = 0; i < n; i++) p[i] = a[i].second;
    c[p[0]] = 0;
  for (int i = 1; i < n; i++) {
    if (a[i].first == a[i - 1].first) {
      c[p[i]] = c[p[i - 1]];
    } else {
      c[p[i]] = c[p[i - 1]] + 1;
    }
  }

  int k = 0;
  while ((1 << k) < n) {
        // k -> k + 1
    for (int i = 0; i < n; i++) {
      p[i] = (p[i] - (1 << k) + n) % n;
    }
    radix_sort(p, c);
    vector<int> c_new(n);
    c_new[p[0]] = 0;
    for (int i = 1; i < n; i++) {
      pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << k)) %
          n]};
      pair<int, int> now = {c[p[i]], c[(p[i] + (1 << k)) % n]};
      if (prev == now) {
        c_new[p[i]] = c_new[p[i - 1]];
      } else {
        c_new[p[i]] = c_new[p[i - 1]] + 1;
      }
    }
    c = c_new;
    k++;
  }

  int q;
  cin >> q;
  while (q--) {
    string t;
    cin >> t;
    int lo = 0, hi = n - 1;
    string ans = "NO\n";

    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      string sub = s.substr(p[mid], min((int)t.size(), n - p[
          mid]));

      if (sub.compare(0, t.size(), t) == 0) {
        ans = "YES\n";
        break;
      } else if (t > sub) {
        lo = mid + 1;
      } else {
        hi = mid - 1;
      }
    }
    cout << ans;
  }
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  int tt;
  tt = 1;
    // cin >> tt;
  while(tt--) {
    SA();
  }
  return 0;
}
```

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: $\mathcal{O}(26N)$

d41d8c, 47 lines

```
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][c]=m+1;  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

```
struct Hashing {
    // 0-base indexing
    int n;
    FenwickTree ft1, ft2;
    Hashing() : n(0), ft1(0), ft2(0) {}
    void build_hash(string &s, int size) {
        init();
        n = size;
        ft1 = FenwickTree(n);
        ft2 = FenwickTree(n);
        for(int i = 0; i < n; i++) {
            int hash_value = p_ip1[i][0]*1LL*s[i] % M1;
            ft1.add(i, hash_value);
            hash_value = p_ip2[i][0]*1LL*s[i] % M2;
            ft2.add(i, hash_value);
```

```
        }
    }
    void update(int i, char c) {
        int hash_value = p_ip1[i][0]*1LL*c % M1;
        ft1.add(i, (-ft1.sum(i, i) + hash_value + M1) % M1);
        hash_value = p_ip2[i][0]*1LL*c % M2;
        ft2.add(i, (-ft2.sum(i, i) + hash_value + M2) % M2);
    }
    array<int, 2> get_hash(int l, int r) {
        array<int, 2> ans;
        ans[0] = ((ft1.sum(l, r) + M1) % M1)*1LL*p_ip1[l][1] %
            M1;
        ans[1] = ((ft2.sum(l, r) + M2) % M2)*1LL*p_ip2[l][1] %
            M2;
        return ans;
    }
    array<int, 2> get_hash() {return get_hash(0, n-1);}
} h, rh;

bool check_palindrome(int i, int j, int n) {
    // 0-base indexing
    return h.get_hash(i, j) == rh.get_hash(n - j - 1, n - i -
        1);
}
```

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 $f(a) < \dots < f(i) \geq \dots \geq f(b)$. 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: $\mathcal{O}(\log(b-a))$

d41d8c, 11 lines

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

d41d8c, 17 lines

```
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: $\mathcal{O}(N\max(w_i))$

d41d8c, 16 lines

```
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 $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i)\le k<hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}((N + (hi - lo))\log N)$

d41d8c, 18 lines

```
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 $[0, 2b)$.

d41d8c, 8 lines

```
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

d41d8c, 10 lines

```
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

d41d8c, 3 lines

```
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

```
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;
}
```