



Tashkent Branch of Moscow State University

# 2men1ramen

Kim Svatoslav, Ashrapov Rafael, Abduraimov Sardor

2024-11-15

- 1 Contest
- 2 Mathematics
- 3 Data structures
- 4 Numerical

5 Number theory

6 Combinatorial

7 Graph

8 Geometry

9 Strings

10 Various

Contest (1)

```
template.cpp34 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;

template<typename T_container, typename T = typename enable_if<
    <!is_same<T_container, string>::value, typename
    T_container::value_type>::type>
ostream& operator<<(ostream &os, const T_container &v) {
    os << '{';
    string sep;
    for (const T &x : v) {
        os << sep << x;
        sep = ", ";
    }
    return os << '}';
}

void dbg_out() { cerr << endl; }
template<typename Head, typename... Tail> void dbg_out(Head H,
    Tail... T) { cerr << ' ' << H; dbg_out(T...); }

#ifdef ONPC
#define deb(...) cerr << '[' << __FILE__ << ':' << __LINE__
    << "]" (" << #__VA_ARGS__ << "):", dbg_out(__VA_ARGS__))
#else
#define deb(...)
#endif

int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
}
```

```
randomizer.h32 lines

/**
 * Author: nor
 * Date: 2024-11-11
 * License: CCO
 * Source: yosupo.judge
 * Description: randomizer
 * Status: works
 */
#pragma once

struct Random : std::mt19937 {
    using std::mt19937::mt19937;
    using std::mt19937::operator();
    static int64_t gen_seed() {
        return std::chrono::steady_clock::now().time_since_epoch().
            count();
    }
    Random() : std::mt19937(gen_seed()) {}
    template <class Int>
    auto operator()(Int a, Int b)
        -> std::enable_if_t<std::is_integral_v<Int>, Int> {
        return std::uniform_int_distribution<Int>(a, b)(*this);
    }
    template <class Int>
    auto operator()(Int a) -> std::enable_if_t<std::is_integral_v
        <Int>, Int> {
        return std::uniform_int_distribution<Int>(0, a - 1)(*this);
    }
    template <class Real>
    auto operator()(Real a, Real b)
        -> std::enable_if_t<std::is_floating_point_v<Real>, Real>
        {
        return std::uniform_real_distribution<Real>(a, b)(*this);
    }
};
```

```
.bashrc3 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 = <
```

```
.vimrc6 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.sh3 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
```

```
troubleshoot.txt52 lines

Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
```

Are you clearing all data structures between test cases?  
Can your algorithm handle the whole range of input?  
Read the full problem statement again.  
Do you handle all corner cases correctly?  
Have you understood the problem correctly?  
Any uninitialized variables?  
Any overflows?  
Confusing N and M, i and j, etc.?  
Are you sure your algorithm works?  
What special cases have you not thought of?  
Are you sure the STL functions you use work as you think?  
Add some assertions, maybe resubmit.  
Create some testcases to run your algorithm on.  
Go through the algorithm for a simple case.  
Go through this list again.  
Explain your algorithm to a teammate.  
Ask the teammate to look at your code.  
Go for a small walk, e.g. to the toilet.  
Is your output format correct? (including whitespace)  
Rewrite your solution from the start or let a teammate do it.

Runtime error:  
Have you tested all corner cases locally?  
Any uninitialized variables?  
Are you reading or writing outside the range of any vector?  
Any assertions that might fail?  
Any possible division by 0? (mod 0 for example)  
Any possible infinite recursion?  
Invalidated pointers or iterators?  
Are you using too much memory?  
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:  
Do you have any possible infinite loops?  
What is the complexity of your algorithm?  
Are you copying a lot of unnecessary data? (References)  
How big is the input and output? (consider scanf)  
Avoid vector, map. (use arrays/unordered\_map)  
What do your teammates think about your algorithm?

Memory limit exceeded:  
What is the max amount of memory your algorithm should need?  
Are you clearing all data structures between test cases?

```
run.sh7 lines

g++ -std=c++17 -Wall -Wextra -pedantic -O2 -Wshadow -Wformat=2
    -Wfloat-equal -Wconversion -Wlogical-op -Wshift-overflow=2
    -Wduplicated-cond -Wcast-qual -Wcast-align -
    D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC -
    D_FORTIFY_SOURCE=2 -fsanitize=address -fsanitize=undefined
    -fno-sanitize-recover -fstack-protector -DONPC main.cpp -
    o main
for file in inp*; do
    if [ -f "$file" ]; then
        cat "$file"
        ./main < "$file"
    fi
done
```

```
stress.sh11 lines

g++ -std=c++17 -O2 smart.cpp -o smart
g++ -std=c++17 -O2 stupid.cpp -o stupid
g++ -std=c++17 -O2 gen.cpp -o gen

for t in $(seq 1 100000); do
    echo "Running test $t"
    ./gen $t > input
```

```
./smart < input > smart_output
./stupid < input > stupid_output
diff smart_output stupid_output || exit
done
```

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  $x=-b/2a$ .

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

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  $a_n=c_1a_{n-1}+\dots+c_ka_{n-k}$ , and  $r_1,\dots,r_k$  are distinct roots of  $x^k-c_1x^{k-1}-\dots-c_k$ , there are  $d_1,\dots,d_k$  s.t.

$$a_n=d_1r_1^n+\dots+d_kr_k^n.$$

Non-distinct roots  $r$  become polynomial factors, e.g.

$$a_n=(d_1n+d_2)r^n.$$

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

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

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

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

Length of median (divides triangle into two equal-area triangles):

$$m_a=\tfrac{1}{2}\sqrt{2b^2+2c^2-a^2}$$

Length of bisector (divides angles in two):

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

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

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

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

2.4.2 Quadrilaterals

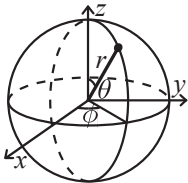
With side lengths  $a,b,c,d$ , diagonals  $e,f$ , diagonals angle  $\theta$ , area  $A$  and magic flux  $F=b^2+d^2-a^2-c^2$ :

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

For cyclic quadrilaterals the sum of opposite angles is  $180^\circ$ ,

$$ef=ac+bd,\text{ and }A=\sqrt{(p-a)(p-b)(p-c)(p-d)}.$$

2.4.3 Spherical coordinates



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

2.5 Sums

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

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

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

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

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

2.6 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let  $X_1,X_2,\dots$  be a sequence of random variables generated by the Markov process. Then there is a transition matrix  $\mathbf{P}=(p_{ij})$ , with  $p_{ij}=\Pr(X_n=i|X_{n-1}=j)$ , and  $\mathbf{p}^{(n)}=\mathbf{P}^n\mathbf{p}^{(0)}$  is the probability distribution for  $X_n$  (i.e.,  $p_i^{(n)}=\Pr(X_n=i)$ ), where  $\mathbf{p}^{(0)}$  is the initial distribution.

$\pi$  is a stationary distribution if  $\pi=\pi\mathbf{P}$ . If the Markov chain is *irreducible* (it is possible to get to any state from any state), then  $\pi_i=\frac{1}{\mathbb{E}(T_i)}$  where  $\mathbb{E}(T_i)$  is the expected time between two visits in state  $i$ .  $\pi_j/\pi_i$  is the expected number of visits in state  $j$  between two visits in state  $i$ .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors,  $\pi_i$  is proportional to node  $i$ 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1).  $\lim_{k\rightarrow\infty}\mathbf{P}^k=1\pi$ .

A Markov chain is an **A-chain** if the states can be partitioned into two sets **A** and **G**, such that all states in **A** are absorbing ( $p_{ii}=1$ ), and all states in **G** leads to an absorbing state in **A**. The probability for absorption in state  $i\in\mathbf{A}$ , when the initial state is  $j$ , is  $a_{ij}=p_{ij}+\sum_{k\in\mathbf{G}}a_{ik}p_{kj}$ . The expected time until absorption, when the initial state is  $i$ , is  $t_i=1+\sum_{k\in\mathbf{G}}p_{ki}t_k$ .

Data structures (3)

OrderStatisticTree.h

**Description:** A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change `null_type`.  
**Time:**  $\mathcal{O}(\log N)$

782797, 16 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
```

```
template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
```

```
void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T< T2 or T> T2, merge t2 into t
}
```

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).  
d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
    const uint64_t C = 114e18 * acos(0) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

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:**  $\mathcal{O}(\log N)$   
0f4bdb, 19 lines

```
struct Tree {
    typedef int T;
    static constexpr T unit = INT_MIN;
    T f(T a, T b) { return max(a, b); } // (any associative fn)
    vector<T> s; int n;
    Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
    void update(int pos, T val) {
        for (s[pos += n] = val; pos /= 2;)
            s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
    }
    T query(int b, int e) { // query [b, e)
        T ra = unit, rb = unit;
        for (b += n, e += n; b < e; b /= 2, e /= 2) {
            if (b % 2) ra = f(ra, s[b++]);
            if (e % 2) rb = f(s[--e], rb);
        }
        return f(ra, rb);
    }
};
```

LazySegmentTree.h  
**Description:** Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.  
**Usage:** Node\* tr = new Node(v, 0, sz(v));  
**Time:**  $\mathcal{O}(\log N)$ .  
"../various/BumpAllocator.h" 34ecf5, 50 lines

```
const int inf = 1e9;
struct Node {
    Node *l = 0, *r = 0;
    int lo, hi, mset = inf, madd = 0, val = -inf;
    Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
    Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
        if (lo + 1 < hi) {
            int mid = lo + (hi - lo)/2;
            l = new Node(v, lo, mid); r = new Node(v, mid, hi);
            val = max(l->val, r->val);
        }
        else val = v[lo];
    }
};
```

```
}
int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
}
void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
        push(), l->set(L, R, x), r->set(L, R, x);
        val = max(l->val, r->val);
    }
}
}
void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
        if (mset != inf) mset += x;
        else madd += x;
        val += x;
    }
    else {
        push(), l->add(L, R, x), r->add(L, R, x);
        val = max(l->val, r->val);
    }
}
}
void push() {
    if (!l) {
        int mid = lo + (hi - lo)/2;
        l = new Node(lo, mid); r = new Node(mid, hi);
    }
    if (mset != inf)
        l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
        l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
}
};
```

SegmentTreeBeats.h  
**Description:** Svatoslav should know all those shit  
**Time:**  $\mathcal{O}(\log N)$   
8d5b12, 180 lines

```
class ExtendedJiDriverSegmentTree {
    static const int64_t INF = 1e15 + 7;

    struct Node {
        int64_t max;
        int maxCnt;
        int64_t secondMax;
        int64_t sum;
        int64_t pushSum;
    };

    vt<Node> tree;
    int n;

    static int leftChild(const int v) {
        return 2 * v;
    }

    static int rightChild(const int v) {
        return 2 * v + 1;
    }

    /* apply min= val to node v */
    void updateWithVal(const int v, const int64_t val) {
        if (tree[v].max > val) {
```

```
            tree[v].sum -= 1LL * (tree[v].max - val) * tree[v].maxCnt;
            tree[v].max = val;
        }
    }

    /* apply += val to node v */
    void doPushSum(const int v, const int l, const int r, const int64_t val) {
        tree[v].max += val;
        tree[v].secondMax += val;
        tree[v].sum += 1LL * (r - l) * val;
        tree[v].pushSum += val;
    }

    /* push lazy update to children of node v */
    void pushToChildren(const int v, const int l, const int r) {
        const int mid = (r + l) / 2;
        doPushSum(leftChild(v), l, mid, tree[v].pushSum);
        doPushSum(rightChild(v), mid, r, tree[v].pushSum);
        tree[v].pushSum = 0;
        updateWithVal(leftChild(v), tree[v].max);
        updateWithVal(rightChild(v), tree[v].max);
    }

    /* pull info from children and recalculate info for v */
    void updateFromChildren(const int v) {
        tree[v].sum = tree[leftChild(v)].sum + tree[rightChild(v)].sum;
        tree[v].max = std::max(tree[leftChild(v)].max, tree[rightChild(v)].max);
        tree[v].secondMax = std::max(tree[leftChild(v)].secondMax, tree[rightChild(v)].secondMax);
        tree[v].maxCnt = 0;
        if (tree[leftChild(v)].max == tree[v].max) {
            tree[v].maxCnt += tree[leftChild(v)].maxCnt;
        } else {
            tree[v].secondMax = std::max(tree[v].secondMax, tree[leftChild(v)].max);
        }
        if (tree[rightChild(v)].max == tree[v].max) {
            tree[v].maxCnt += tree[rightChild(v)].maxCnt;
        } else {
            tree[v].secondMax = std::max(tree[v].secondMax, tree[rightChild(v)].max);
        }
    }

    /* build segment tree according to inputArray */
    void build(const int v, const int l, const int r, const vt<int64_t>& inputArray) {
        tree[v].pushSum = 0;
        if (l + 1 == r) {
            tree[v].max = inputArray[l];
            tree[v].secondMax = -INF;
            tree[v].maxCnt = 1;
            tree[v].sum = inputArray[l];
        } else {
            const int mid = (r + l) / 2;
            build(leftChild(v), l, mid, inputArray);
            build(rightChild(v), mid, r, inputArray);
            updateFromChildren(v);
        }
    }

    /* min= val on segment [ql, qr) */
    void updateMinEq(const int v, const int l, const int r, const int ql, const int qr, const int64_t val) {
        if (qr <= l || r <= ql || tree[v].max <= val) {
```

```
        return;
    }
    if (ql <= l && r <= qr && tree[v].secondMax < val) {
        updateWithVal(v, val);
        return;
    }
    pushToChildren(v, l, r);
    const int mid = (r + l) / 2;
    updateMinEq(leftChild(v), l, mid, ql, qr, val);
    updateMinEq(rightChild(v), mid, r, ql, qr, val);
    updateFromChildren(v);
}

/* += val on segment [ql, qr) */
void updatePlusEq(const int v, const int l, const int r,
    const int ql, const int qr, const int64_t val) {
    if (qr <= l || r <= ql) {
        return;
    }
    if (ql <= l && r <= qr) {
        doPushSum(v, l, r, val);
        return;
    }
    pushToChildren(v, l, r);
    const int mid = (r + l) / 2;
    updatePlusEq(leftChild(v), l, mid, ql, qr, val);
    updatePlusEq(rightChild(v), mid, r, ql, qr, val);
    updateFromChildren(v);
}

/* find sum on segment [ql, qr) */
int64_t findSum(const int v, const int l, const int r, const
    int ql, const int qr) {
    if (qr <= l || r <= ql) {
        return 0;
    }
    if (ql <= l && r <= qr) {
        return tree[v].sum;
    }
    pushToChildren(v, l, r);
    const int mid = (r + l) / 2;
    return findSum(leftChild(v), l, mid, ql, qr) + findSum(
        rightChild(v), mid, r, ql, qr);
}

public:
ExtendedJiDriverSegmentTree(const vt<int64_t>& inputArray) {
    n = inputArray.size();
    tree.resize(4 * n);
    build(1, 0, n, inputArray);
}

/* min= val on segment [ql, qr) */
void updateMinEq(const int ql, const int qr, const int val) {
    updateMinEq(1, 0, n, ql, qr, val);
}

/* += val on segment [ql, qr) */
void updatePlusEq(const int ql, const int qr, const int val)
    {
        updatePlusEq(1, 0, n, ql, qr, val);
    }

/* find sum on segment [ql, qr) */
int64_t findSum(const int ql, const int qr) {
    return findSum(1, 0, n, ql, qr);
}
};
```

```
/*
int main() {
    int n;
    std::cin >> n;
    vt<int64_t> inputArray(n);
    for (int64_t Eval : inputArray) {
        std::cin >> val;
    }
    ExtendedJiDriverSegmentTree segTree(inputArray);
    int q;
    std::cin >> q;
    for (int i = 0; i < q; i++) {
        int type, ql, qr;
        std::cin >> type >> ql >> qr;
        ql--;
        if (type == 1) {
            int val;
            std::cin >> val;
            segTree.updateMinEq(ql, qr, val);
        } else if (type == 2) {
            int toAdd;
            std::cin >> toAdd;
            segTree.updatePlusEq(ql, qr, toAdd);
        } else {
            std::cout << segTree.findSum(ql, qr) << '\n';
        }
    }
    return 0;
}
*/
```

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

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

LineContainer.h  
Description: Container where you can add lines of the form  $kx+m$ , and query maximum values at points  $x$ . Useful for dynamic programming (“convex hull trick”).

```
Time:  $\mathcal{O}(\log N)$ 
Sec1c7, 30 lines

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

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

```
9556fc, 55 lines

struct Node {
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int val) : val(val), y(rand()) {}
    void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};
    if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
        auto pa = split(n->l, k);
        n->l = pa.second;
        n->recalc();
        return {pa.first, n};
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = pa.first;
        n->recalc();
        return {n, pa.second};
    }
}

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
```

```
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        l->recalc();
        return l;
    } else {
        r->l = merge(l, r->l);
        r->recalc();
        return r;
    }
}
```

```
Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second);
}
```

```
// Example application: move the range [l, r) to index k
void move(Node&& t, int l, int r, int k) {
    Node *a, *b, *c;
    tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}
```

FenwickTree.h  
**Description:** Computes partial sums  $a[0] + a[1] + \dots + a[pos - 1]$ , and updates single elements  $a[i]$ , taking the difference between the old and new value.  
**Time:** Both operations are  $\mathcal{O}(\log N)$ .

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

FenwickTree2d.h  
**Description:** Computes sums  $a[i,j]$  for all  $i < j$ , and increases single elements  $a[i,j]$ . Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).  
**Time:**  $\mathcal{O}(\log^2 N)$ . (Use persistent segment trees for  $\mathcal{O}(\log N)$ .)

```
"FenwickTree.h"
157f07, 22 lines

struct FT2 {
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
    }
    void init() {
        for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
    }
    int ind(int x, int y) {
```

```
        return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) {
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum;
    }
};
```

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

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 + 1);
            rep(j, 0, sz(jmp[k]))
                jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
        }
    }
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
};
```

## Numerical (4)

### 4.1 Polynomials and recurrences

```
Polynomial.h
c9b7b0, 17 lines

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

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

```
"Polynomial.h"
b00bfe, 23 lines

vector<double> polyRoots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
```

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

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

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

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

```
".../number-theory/ModPow.h"
96548b, 20 lines

vector<ll> berlekampMassey(vector<ll> s) {
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1;
    rep(i, 0, n) { ++m;
        ll d = s[i] % mod;
        rep(j, 1, L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod;
        rep(j, m, n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }

    C.resize(L + 1); C.erase(C.begin());
    for (ll& x : C) x = (mod - x) % mod;
```

```
    return C;
}
```

### LinearRecurrence.h

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

f4e444, 26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
        return res;
    };

    Poly pol(n + 1), e(pol);
    pol[0] = e[1] = 1;

    for (++k; k; k /= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    }

    ll res = 0;
    rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
    return res;
}
```

## 4.2 Optimization

### GoldenSectionSearch.h

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

31d45b, 14 lines

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

### Integrate.h

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

4756fc, 7 lines

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

### IntegrateAdaptive.h

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

92dd79, 15 lines

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

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

template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
    return rec(f, a, b, eps, S(a, b));
}
```

### Simplex.h

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

aa8530, 68 lines

```
typedef double T; // long double, Rational, double + modP>...
typedef vector<T> vd;
typedef vector<vvd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
    int m, n;
    vi N, B;
    vvd D;

    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
        rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
        rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
        rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m+1][n] = 1;
    }

    void pivot(int r, int s) {
        T *a = D[r].data(), inv = 1 / a[s];
        rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
```

```
            T *b = D[i].data(), inv2 = b[s] * inv;
            rep(j,0,n+2) b[j] -= a[j] * inv2;
            b[s] = a[s] * inv2;
        }
        rep(j,0,n+2) if (j != s) D[r][j] *= inv;
        rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }

    bool simplex(int phase) {
        int x = m + phase - 1;
        for (;;) {
            int s = -1;
            rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
            if (D[x][s] >= -eps) return true;
            int r = -1;
            rep(i,0,m) {
                if (D[i][s] <= eps) continue;
                if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                    < MP(D[r][n+1] / D[r][s], B[r])) r = i;
            }
            if (r == -1) return false;
            pivot(r, s);
        }
    }

    T solve(vd &x) {
        int r = 0;
        rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
        if (D[r][n+1] < -eps) {
            pivot(r, n);
            if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
            rep(i,0,m) if (B[i] == -1) {
                int s = 0;
                rep(j,1,n+1) ltj(D[i]);
                pivot(i, s);
            }
        }
        bool ok = simplex(1); x = vd(n);
        rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
        return ok ? D[m][n+1] : inf;
    }
};
```

## 4.3 Matrices

### Determinant.h

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

bd5cec, 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;
}
```

### IntDeterminant.h

**Description:** Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

**Time:**  $\mathcal{O}(N^3)$

```
3313dc, 18 lines

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

### SolveLinear.h

**Description:** Solves  $A * x = b$ . If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in  $A$  and  $b$  is lost.

**Time:**  $\mathcal{O}(n^2m)$

```
44c9ab, 38 lines

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

int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
            rep(j,i,n) if (fabs(b[j]) > eps) return -1;
            break;
        }
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) swap(A[j][i], A[j][bc]);
        bv = 1/A[i][i];
        rep(j,i+1,n) {
            double fac = A[j][i] * bv;
            b[j] -= fac * b[i];
            rep(k,i+1,m) A[j][k] -= fac*A[i][k];
        }
        rank++;
    }

    x.assign(m, 0);
    for (int i = rank; i--;) {
        b[i] /= A[i][i];
        x[col[i]] = b[i];
        rep(j,0,i) b[j] -= A[j][i] * b[i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

### SolveLinear2.h

**Description:** To get all uniquely determined values of  $x$  back from SolveLinear, make the following changes:

```
08e495, 7 lines

"SolveLinear.h"

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

### SolveLinearBinary.h

**Description:** Solves  $Ax = b$  over  $\mathbb{F}_2$ . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys  $A$  and  $b$ .

**Time:**  $\mathcal{O}(n^2m)$

```
fa2d7a, 34 lines

typedef bitset<1000> bs;

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

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
        rep(j,0,i) b[j] ^= A[j][i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

### MatrixInverse.h

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

**Time:**  $\mathcal{O}(n^3)$

```
ebfff6, 35 lines

int matInv(vector<vector<double>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) {
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
            if (fabs(A[j][k]) > fabs(A[r][c]))
                r = j, c = k;
```

```
if (fabs(A[r][c]) < 1e-12) return i;
A[i].swap(A[r]); tmp[i].swap(tmp[r]);
rep(j,0,n)
    swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
swap(col[i], col[c]);
double v = A[i][i];
rep(j,i+1,n) {
    double f = A[j][i] / v;
    A[j][i] = 0;
    rep(k,i+1,n) A[j][k] -= f*A[i][k];
    rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
}
rep(j,i+1,n) A[i][j] /= v;
rep(j,0,n) tmp[i][j] /= v;
A[i][i] = 1;
}

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
}
```

```
rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
return n;
}
```

### Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.

If  $|d_i| > |p_i| + |q_{i-1}|$  for all  $i$ , or  $|d_i| > |p_{i-1}| + |q_i|$ , or the matrix is positive definite, the algorithm is numerically stable and neither tr nor the check for  $\text{diag}[i] == 0$  is needed.

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

```
8f9fa8, 26 lines

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



```
        b[i] /= diag[i];
        if (i) b[i-1] -= b[i]*super[i-1];
    }
}
return b;
}
```

## 4.4 Fourier transforms

**FastFourierTransform.h**  
**Description:**  $\text{fft}(a)$  computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution:  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by  $n$ , reverse(start+1, end), FFT back. Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFTMod.  
**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1s$  for  $N = 2^{22}$ )

<pre>typedef complex&lt;double&gt; C; typedef vector&lt;double&gt; vd; void fft(vector&lt;C&gt;&amp; a) {     int n = sz(a), L = 31 - __builtin_clz(n);     static vector&lt;complex&lt;long double&gt;&gt; R(2, 1);     static vector&lt;C&gt; rt(2, 1); // (^ 10% faster if double)     for (static int k = 2; k &lt; n; k *= 2) {         R.resize(n); rt.resize(n);         auto x = polar(1.0L, acos(-1.0L) / k);         rep(i,k,2*k) rt[i] = R[i] = i&amp;1 ? R[i/2] * x : R[i/2];     }     vi rev(n);     rep(i,0,n) rev[i] = (rev[i / 2]   (i &amp; 1) &lt;&lt; L) / 2;     rep(i,0,n) if (i &lt; rev[i]) swap(a[i], a[rev[i]]);     for (int k = 1; k &lt; n; k *= 2)         for (int i = 0; i &lt; n; i += 2 * k) rep(j,0,k) {             C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)             a[i + j + k] = a[i + j] - z;             a[i + j] += z;         } } vd conv(const vd&amp; a, const vd&amp; b) {     if (a.empty()    b.empty()) return {};     vd res(sz(a) + sz(b) - 1);     int L = 32 - __builtin_clz(sz(res)), n = 1 &lt;&lt; L;     vector&lt;C&gt; in(n), out(n);     copy(all(a), begin(in));     rep(i,0,sz(b)) in[i].imag(b[i]);     fft(in);     for (C&amp; x : in) x *= x;     rep(i,0,n) out[i] = in[-i &amp; (n - 1)] - conj(in[i]);     fft(out);     rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);     return res; }</pre>	00ced6, 35 lines
---	------------------

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

<pre>"FastFourierTransform.h" typedef vector&lt;ll&gt; vl; template&lt;int M&gt; vl convMod(const vl &amp;a, const vl &amp;b) {     if (a.empty()    b.empty()) return {};     vl res(sz(a) + sz(b) - 1);     int B=32-__builtin_clz(sz(res)), n=1&lt;&lt;B, cut=int(sqrt(M));     vector&lt;C&gt; L(n), R(n), outs(n), outl(n);     rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);     rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);     fft(L), fft(R);</pre>	b82773, 22 lines
--	------------------

<pre>rep(i,0,n) {     int j = -i &amp; (n - 1);     outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);     outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i; } fft(outl), fft(outs); rep(i,0,sz(res)) {     ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);     ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);     res[i] = ((av % M * cut + bv) % M * cut + cv) % M; } return res; }</pre>	
--	--

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

<pre>"../number-theory/ModPow.h" const ll mod = (119 &lt;&lt; 23) + 1, root = 62; // = 998244353 // For p &lt; 2^30 there is also e.g. 5 &lt;&lt; 25, 7 &lt;&lt; 26, 479 &lt;&lt; 21 // and 483 &lt;&lt; 21 (same root). The last two are &gt; 10^9. typedef vector&lt;ll&gt; vl; void ntt(vl &amp;a) {     int n = sz(a), L = 31 - __builtin_clz(n);     static vl rt(2, 1);     for (static int k = 2, s = 2; k &lt; n; k *= 2, s++) {         rt.resize(n);         ll z[] = {1, modpow(root, mod &gt;&gt; s)};         rep(i,k,2*k) rt[i] = rt[i / 2] * z[i &amp; 1] % mod;     }     vi rev(n);     rep(i,0,n) rev[i] = (rev[i / 2]   (i &amp; 1) &lt;&lt; L) / 2;     rep(i,0,n) if (i &lt; rev[i]) swap(a[i], a[rev[i]]);     for (int k = 1; k &lt; n; k *= 2)         for (int i = 0; i &lt; n; i += 2 * k) rep(j,0,k) {             ll z = rt[j + k] * a[i + j + k] % mod, &amp;ai = a[i + j];             a[i + j + k] = ai - z + (z &gt; ai ? mod : 0);             ai += (ai + z &gt;= mod ? z - mod : z);         } } vl conv(const vl &amp;a, const vl &amp;b) {     if (a.empty()    b.empty()) return {};     int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),         n = 1 &lt;&lt; B;     int inv = modpow(n, mod - 2);     vl L(a), R(b), out(n);     L.resize(n), R.resize(n);     ntt(L), ntt(R);     rep(i,0,n)         out[-i &amp; (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;     ntt(out);     return {out.begin(), out.begin() + s}; }</pre>	ced03d, 35 lines
--	------------------

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

<pre>void FST(vi&amp; a, bool inv) {     for (int n = sz(a), step = 1; step &lt; n; step *= 2) {</pre>	464cf3, 16 lines
--	------------------

<pre>    for (int i = 0; i &lt; n; i += 2 * step) rep(j,i,i+step) {         int &amp;u = a[j], &amp;v = a[j + step]; tie(u, v) =             inv ? pii(v - u, u) : pii(v, u + v); // AND             inv ? pii(v, u - v) : pii(u + v, u); // OR             pii(u + v, u - v); // XOR         }     }     if (inv) for (int&amp; x : a) x /= sz(a); // XOR only } vi conv(vi a, vi b) {     FST(a, 0); FST(b, 0);     rep(i,0,sz(a)) a[i] *= b[i];     FST(a, 1); return a; }</pre>	
---	--

## Number theory (5)

### 5.1 Modular arithmetic

**ModularArithmetic.h**  
**Description:** Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

<pre>"euclid.h" const ll mod = 17; // change to something else struct Mod {     ll x;     Mod(ll xx) : x(xx) {}     Mod operator+(Mod b) { return Mod((x + b.x) % mod); }     Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }     Mod operator*(Mod b) { return Mod((x * b.x) % mod); }     Mod operator/(Mod b) { return *this * invert(b); }     Mod invert(Mod a) {         ll x, y, g = euclid(a.x, mod, x, y);         assert(g == 1); return Mod((x + mod) % mod);     }     Mod operator^(ll e) {         if (!e) return Mod(1);         Mod r = *this ^ (e / 2); r = r * r;         return e&amp;1 ? *this * r : r;     } };</pre>	35bfea, 18 lines
--	------------------

**ModInverse.h**  
**Description:** Pre-computation of modular inverses. Assumes  $\text{LIM} \leq \text{mod}$  and that mod is a prime.

<pre>const ll mod = 1000000007, LIM = 200000; ll* inv = new ll[LIM] - 1; inv[1] = 1; rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;</pre>	6f684f, 3 lines
--	-----------------

<pre>const ll mod = 1000000007; // faster if const</pre>	b83e45, 8 lines
--	-----------------

<pre>ll modpow(ll b, ll e) {     ll ans = 1;     for (; e; b = b * b % mod, e /= 2)         if (e &amp; 1) ans = ans * b % mod;     return ans; }</pre>	
---	--

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

<pre>ll modLog(ll a, ll b, ll m) {     ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;</pre>	c040b8, 11 lines
--	------------------

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

ModSum.h

**Description:** Sums of mod'ed arithmetic progressions.

modsum(to, c, k, m) =  $\sum_{i=0}^{to-1} (ki + c) \% m$ . divsum is similar but for floored division.

**Time:**  $\log(m)$ , with a large constant.

5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
```

```
ull divsum(ull to, ull c, ull k, ull m) {
    ull res = k / m * sumsq(to) + c / m * to;
    k %= m; c %= m;
    if (!k) return res;
    ull to2 = (to * k + c) / m;
    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}
```

```
ll modsum(ull to, ll c, ll k, ll m) {
    c = ((c % m) + m) % m;
    k = ((k % m) + m) % m;
    return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

**Description:** Calculate  $a \cdot b \bmod c$  (or  $a^b \bmod c$ ) for  $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$ .

**Time:**  $\mathcal{O}(1)$  for modmul,  $\mathcal{O}(\log b)$  for modpow

bbbd8f, 11 lines

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

ModSqrt.h

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

**Time:**  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$

"ModPow.h"19a793, 24 lines

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

```
        for (m = 0; m < r && t != 1; ++m)
            t = t * t % p;
        if (m == 0) return x;
        ll gs = modpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
    }
}
```

DiophantineEquation.h

**Description:** Solving linear diophantine equations  $ax + by = c$ , how many of them

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

467986, 57 lines

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

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

```
void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}
```

```
int find_all_solutions(int a, int b, int c, int minx, int maxx,
    int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g, b /= g;
    int sign_a = a > 0 ? +1 : -1, sign_b = b > 0 ? +1 : -1;
    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;
    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;
```

```
    if (lx2 > rx2) swap(lx2, rx2);
    int lx = max(lx1, lx2), rx = min(rx1, rx2);
    if (lx > rx) return 0;
    return (rx - lx) / abs(b) + 1;
}
```

5.2 Primality

FastEratosthenes.h

**Description:** Prime sieve for generating all primes smaller than LIM.

**Time:** LIM=1e9  $\approx$  1.5s

6b2912, 20 lines

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

MillerRabin.h

**Description:** Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to  $7 \cdot 10^{18}$ ; for larger numbers, use Python and extend A randomly.

**Time:** 7 times the complexity of  $a^b \bmod c$ .

"ModMulLL.h"60dcd1, 12 lines

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

Factor.h

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

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

"ModMulLL.h", "MillerRabin.h"d8d98d, 18 lines

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

```
    if (n == 1) return {};  
    if (isPrime(n)) return {n};  
    ull x = pollard(n);  
    auto l = factor(x), r = factor(n / x);  
    l.insert(l.end(), all(r));  
    return l;  
}
```

### 5.3 Divisibility

CRT.h

**Description:** Chinese Remainder Theorem.  
 $x \equiv a_i \bmod{m_i}$   $m_i$  is co-prime  
**Time:**  $\log(n)$

euclid.h8bfc7e, 18 lines

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

### 5.4 Fractions

ContinuedFractions.h

**Description:** Given  $N$  and a real number  $x \geq 0$ , finds the closest rational approximation  $p/q$  with  $p, q \leq N$ . It will obey  $|p/q - x| \leq 1/qN$ .  
For consecutive convergents,  $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$ . ( $p_k/q_k$  alternates between  $> x$  and  $< x$ .) If  $x$  is rational,  $y$  eventually becomes  $\infty$ ; if  $x$  is the root of a degree 2 polynomial the  $a$ 's eventually become cyclic.  
**Time:**  $\mathcal{O}(\log N)$

dd6c5e, 21 lines

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

FracBinarySearch.h

**Description:** Given  $f$  and  $N$ , finds the smallest fraction  $p/q \in [0, 1]$  such that  $f(p/q)$  is true, and  $p, q \leq N$ . You may want to throw an exception from  $f$  if it finds an exact solution, in which case  $N$  can be removed.  
**Usage:** fracBS({}(Frac f) { return f.p>=3\*f.q; }, 10); // {1,3}  
**Time:**  $\mathcal{O}(\log(N))$

27ab3e, 25 lines

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

### 5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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 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:

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

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

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

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

044568, 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.

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

#### 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	$\sim 2\text{e}5$	$\sim 2\text{e}8$

### 6.2.2 Lucas’ Theorem

Let  $n,m$  be non-negative integers and  $p$  a prime. Write  $n=n_kp^k+...+n_1p+n_0$  and  $m=m_kp^k+...+m_1p+m_0$ . Then  $\binom{n}{m}\equiv\prod_{i=0}^k\binom{n_i}{m_i}\pmod p$ .

### 6.2.3 Binomials

multinomial.h

**Description:** Computes  $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n}=\frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$ . a0a312, 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).

$$B[0,\ldots]=[1,-\tfrac{1}{2},\tfrac{1}{6},0,-\tfrac{1}{30},0,\tfrac{1}{42},\ldots]$$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

### 6.3.2 Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

$$c(n,k)=c(n-1,k-1)+(n-1)c(n-1,k), \; c(0,0)=1$$

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

$$c(8,k)=8,0,5040,13068,13132,6769,1960,322,28,1$$

$$c(n,2)=0,0,1,3,11,50,274,1764,13068,109584,\ldots$$

### 6.3.3 Eulerian numbers

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

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

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

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

### 6.3.4 Stirling numbers of the second kind

Partitions of  $n$  distinct elements into exactly  $k$  groups.

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

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

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

### 6.3.5 Bell numbers

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

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

### 6.3.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$

# on  $k$  existing trees of size  $n_i$ :  $n_1n_2\cdots n_kn^{k-2}$

# with degrees  $d_i$ :  $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

### 6.3.7 Catalan numbers

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

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

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

- 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:**  $\mathcal{O}(VE)$

830a8f, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
    nodes[s].dist = 0;
    sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

    int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
    rep(i,0,lim) for (Ed ed : eds) {
        Node cur = nodes[ed.a], &dest = nodes[ed.b];
        if (abs(cur.dist) == inf) continue;
        ll d = cur.dist + ed.w;
        if (d < dest.dist) {
            dest.prev = ed.a;
            dest.dist = (i < lim-1 ? d : -inf);
        }
    }
    rep(i,0,lim) for (Ed e : eds) {
        if (nodes[e.a].dist == -inf)
            nodes[e.b].dist = -inf;
    }
}
```

### FloydWarshall.h

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

**Time:**  $\mathcal{O}(N^3)$

531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
    int n = sz(m);
    rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
    rep(k,0,n) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) {
            auto newDist = max(m[i][k] + m[k][j], -inf);
            m[i][j] = min(m[i][j], newDist);
        }
    rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

### TopoSort.h

**Description:** Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than  $n$  – nodes reachable from cycles will not be returned.

**Time:**  $\mathcal{O}(|V|+|E|)$

66a137, 14 lines

```
vi topoSort(const vector<vi>& gr) {
    vi indeg(sz(gr)), ret;
    for (auto& li : gr) for (int x : li) indeg[x]++;
    queue<int> q; // use priority-queue for lexic. largest ans.
    rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
    while (!q.empty()) {
        int i = q.front(); // top() for priority queue
        ret.push_back(i);
        q.pop();
        for (int x : gr[i])
            if (--indeg[x] == 0) q.push(x);
    }
}
```

```
    return ret;
}
```

## 7.2 Network flow

### PushRelabel.h

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

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

```
struct PushRelabel {
    struct Edge {
        int dest, back;
        ll f, c;
    };
    vector<vector<Edge>> g;
    vector<ll> ec;
    vector<Edge*> cur;
    vector<vi> hs; vi H;
    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

    void addEdge(int s, int t, ll cap, ll rcap=0) {
        if (s == t) return;
        g[s].push_back({t, sz(g[t]), 0, cap});
        g[t].push_back({s, sz(g[s])-1, 0, rcap});
    }

    void addFlow(Edge& e, ll f) {
        Edge &back = g[e.dest][e.back];
        if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
        e.f += f; e.c -= f; ec[e.dest] += f;
        back.f -= f; back.c += f; ec[back.dest] -= f;
    }

    ll calc(int s, int t) {
        int v = sz(g); H[s] = v; ec[t] = 1;
        vi co(2*v); co[0] = v-1;
        rep(i,0,v) cur[i] = g[i].data();
        for (Edge& e : g[s]) addFlow(e, e.c);

        for (int hi = 0;;) {
            while (hs[hi].empty()) if (!hi--) return -ec[s];
            int u = hs[hi].back(); hs[hi].pop_back();
            while (ec[u] > 0) // discharge u
                if (cur[u] == g[u].data() + sz(g[u])) {
                    H[u] = 1e9;
                    for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
                        H[u] = H[e.dest]+1, cur[u] = &e;
                    if (++co[H[u]],!--co[hi] && hi < v)
                        rep(i,0,v) if (hi < H[i] && H[i] < v)
                            --co[H[i]], H[i] = v + 1;
                    hi = H[u];
                } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
                    addFlow(*cur[u], min(ec[u], cur[u]->c));
                else ++cur[u];
        }

        bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

### MinCostMaxFlow.h

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

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

```
#include <bits/extc++.h>

const ll INF = numeric_limits<ll>::max() / 4;
```

```
struct MCMF {
    struct edge {
        int from, to, rev;
        ll cap, cost, flow;
    };
    int N;
    vector<vector<edge>> ed;
    vi seen;
    vector<ll> dist, pi;
    vector<edge*> par;

    MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

    void addEdge(int from, int to, ll cap, ll cost) {
        if (from == to) return;
        ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
        ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
    }

    void path(int s) {
        fill(all(seen), 0);
        fill(all(dist), INF);
        dist[s] = 0; ll di;

        __gnu_pbds::priority_queue<pair<ll, int>> q;
        vector<decltype(q)::point_iterator> its(N);
        q.push({ 0, s });

        while (!q.empty()) {
            s = q.top().second; q.pop();
            seen[s] = 1; di = dist[s] + pi[s];
            for (edge& e : ed[s]) if (!seen[e.to]) {
                ll val = di - pi[e.to] + e.cost;
                if (e.cap - e.flow > 0 && val < dist[e.to]) {
                    dist[e.to] = val;
                    par[e.to] = &e;
                    if (its[e.to] == q.end())
                        its[e.to] = q.push({ -dist[e.to], e.to });
                    else
                        q.modify(its[e.to], { -dist[e.to], e.to });
                }
            }
        }
        rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
    }

    pair<ll, ll> maxflow(int s, int t) {
        ll totflow = 0, totcost = 0;
        while (path(s), seen[t]) {
            ll fl = INF;
            for (edge* x = par[t]; x; x = par[x->from])
                fl = min(fl, x->cap - x->flow);

            totflow += fl;
            for (edge* x = par[t]; x; x = par[x->from]) {
                x->flow += fl;
                ed[x->to][x->rev].flow -= fl;
            }
        }
        rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
        return {totflow, totcost/2};
    }
};
```

```
// If some costs can be negative, call this before maxflow:
void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
```

```
        rep(i,0,N) if (pi[i] != INF)
            for (edge& e : ed[i]) if (e.cap)
                if ((v = pi[i] + e.cost) < pi[e.to])
                    pi[e.to] = v, ch = 1;
        assert(it >= 0); // negative cost cycle
    }
};
```

### EdmondsKarp.h

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

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

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

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

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

### MinCut.h

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

### GlobalMinCut.h

**Description:** Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

**Time:**  $\mathcal{O}(V^3)$

```
pair<int, vi> globalMinCut(vector<vi> mat) {
    pair<int, vi> best = {INT_MAX, {}};
    int n = sz(mat);
    vector<vi> co(n);
    rep(i,0,n) co[i] = {i};
    rep(ph,1,n) {
        vi w = mat[0];
        size_t s = 0, t = 0;
```

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

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

"PushRelabel.h"0418b3, 13 lines

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

### 7.3 Matching

DFSMatching.h  
Description: Fast bipatrite algorithm  
Time:  $\mathcal{O}(VE)$ , in real better

"randomizer.h"c58da3, 70 lines

```
template <bool ToShuffle = false>
struct bipartite_matching {
    int n, m, flow = 0;
    vt<vt<int>>> g;
    vt<int> matchL, matchR, dist;
    bipartite_matching(int _n, int _m): n(_n), m(_m), g(_n),
        matchL(_n, -1), matchR(_m, -1), dist(_n) {}
    void add(int u, int v) { g[u].push_back(v); }
    void bfs() {
        std::queue<int> q;
        for (int u = 0; u < n; ++u) {
            if (!matchL[u]) q.push(u), dist[u] = 0;
            else dist[u] = -1;
        }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto v : g[u]) if (~matchR[v] && !dist[matchR[v]])
                {
                    dist[matchR[v]] = dist[u] + 1;
                    q.push(matchR[v]);
                }
        }
    }
    bool dfs(int u) {
        for (auto v : g[u])
            if (!matchR[v]) {
                matchL[u] = v, matchR[v] = u;
                return true;
            }
    }
}
```

```

    }
    for (auto v : g[u])
        if (dist[matchR[v]] == dist[u] + 1 &&
            dfs(matchR[v])) {
            matchL[u] = v, matchR[v] = u;
            return true;
        }
    return false;
}
int get_max_matching() {
    if constexpr (ToShuffle) {
        Random rng;
        for (int i = 0; i < n; ++i)
            std::random_shuffle(std::begin(g[i]), std::end(g[i]),
                                rng);
    }
    while (true) {
        bfs();
        int augment = 0;
        for (int u = 0; u < n; ++u)
            if (!matchL[u]) augment += dfs(u);
        if (!augment) break;
        flow += augment;
    }
    return flow;
}
std::pair<vt<int>, vt<int>> minimum_vertex_cover() {
    vt<int> L, R;
    for (int u = 0; u < n; ++u) {
        if (!dist[u])
            L.push_back(u);
        else if (~matchL[u])
            R.push_back(matchL[u]);
    }
    return {L, R};
}
vt<std::pair<int, int>> get_edges() {
    vt<std::pair<int, int>> ans;
    for (int u = 0; u < n; ++u)
        if (matchL[u] != -1)
            ans.emplace_back(u, matchL[u]);
    return ans;
}
}
```

WeightedMatching.h  
Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires  $N \leq M$ .  
Time:  $\mathcal{O}(N^2M)$

1e0fe9, 31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) {
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i,1,n) {
        p[0] = i;
        int j0 = 0; // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true;
            int i0 = p[j0], j1, delta = INT_MAX;
            rep(j,1,m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;
            }
        } while (j1 > 0);
        p[j1] = i0;
        if (j1 > 0) ans[i0 - 1] = j1;
    }
    return {ans, p};
}
```

```

    }
    rep(j,0,m) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
    }
    j0 = j1;
} while (p[j0]);
while (j0) { // update alternating path
    int j1 = pre[j0];
    p[j0] = p[j1], j0 = j1;
}
}
rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
return {-v[0], ans}; // min cost
}
```

GeneralMatching.h  
Description: Matching for general graphs. Fails with probability  $N/mod$ .  
Time:  $\mathcal{O}(N^3)$

"../numerical/MatrixInverse-mod.h"cb1912, 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
    vector<vector<ll>> mat(N, vector<ll>(N)), A;
    for (pii pa : ed) {
        int a = pa.first, b = pa.second, r = rand() % mod;
        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
    }

    int r = matInv(A = mat), M = 2*N - r, fi, fj;
    assert(r % 2 == 0);

    if (M != N) do {
        mat.resize(M, vector<ll>(M));
        rep(i,0,N) {
            mat[i].resize(M);
            rep(j,N,M) {
                int r = rand() % mod;
                mat[i][j] = r, mat[j][i] = (mod - r) % mod;
            }
        } while (matInv(A = mat) != M);

    vi has(M, 1); vector<pii> ret;
    rep(it,0,M/2) {
        rep(i,0,M) if (has[i])
            rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
                fi = i; fj = j; goto done;
            } assert(0); done:
        if (fj < N) ret.emplace_back(fi, fj);
        has[fi] = has[fj] = 0;
        rep(sw,0,2) {
            ll a = modpow(A[fi][fj], mod-2);
            rep(i,0,M) if (has[i] && A[i][fj]) {
                ll b = A[i][fj] * a % mod;
                rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
            }
            swap(fi,fj);
        }
    }
    return ret;
}
```

### 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:**  $\mathcal{O}(E + V)$

97109c, 59 lines

```
vector<bool> visited; // keeps track of which vertices are
                      already visited

// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector when
// dfs leaves it.
void dfs(int v, vector<vector<int>> const& adj, vector<int> &
output) {
    visited[v] = true;
    for (auto u : adj[v])
        if (!visited[u])
            dfs(u, adj, output);
    output.push_back(v);
}

// input: adj — adjacency list of G
// output: components — the strongly connected components in G
// output: adj_cond — adjacency list of G^SCC (by root
// vertices)
void strongly_connected_components(vector<vector<int>> const&
adj,
                                vector<vector<int>> &
                                components,
                                vector<vector<int>> &adj_cond
                                ) {

    int n = adj.size();
    components.clear(), adj_cond.clear();

    vector<int> order; // will be a sorted list of G's vertices
                       by exit time

    visited.assign(n, false);

    // first series of depth first searches
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i, adj, order);

    // create adjacency list of G^T
    vector<vector<int>> adj_rev(n);
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            adj_rev[u].push_back(v);

    visited.assign(n, false);
    reverse(order.begin(), order.end());

    vector<int> roots(n, 0); // gives the root vertex of a vertex
                             's SCC

    // second series of depth first searches
    for (auto v : order)
        if (!visited[v]) {
            vt<int> component;
            dfs(v, adj_rev, component);
            components.push_back(component);
            int root = *min_element(begin(component), end(component))
            ;
            for (auto u : component)
                roots[u] = root;
        }

    // add edges to condensation graph
```

BiconnectedComponents 2sat EulerWalk

```
adj_cond.assign(n, {});
for (int v = 0; v < n; v++)
    for (auto u : adj[v])
        if (roots[v] != roots[u])
            adj_cond[roots[v]].push_back(roots[u]);
}
```

BiconnectedComponents.h

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

c6b7c7, 32 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, top = me;
    for (auto [y, e] : ed[at]) if (e != par) {
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);
            if (up == me) {
                st.push_back(e);
                f(vi(st.begin() + si, st.end()));
                st.resize(si);
            }
            else if (up < me) st.push_back(e);
            else { /* e is a bridge */ }
        }
    }
    return top;
}

template<class F>
void bicomps(F f) {
    num.assign(sz(ed), 0);
    rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}

2sat.h
Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type (a||b)&&(!a||c)&&(d||b)&&... becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~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.
```

5f9706, 56 lines

```
struct TwoSat {
    int N;
```

```
vector<vi> gr;
vi values; // 0 = false, 1 = true

TwoSat(int n = 0) : N(n), gr(2*n) {}

int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
}

void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
}

void setValue(int x) { either(x, x); }

void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
        int next = addVar();
        either(cur, ~li[i]);
        either(cur, next);
        either(~li[i], next);
        cur = ~next;
    }
    either(cur, ~li[1]);
}

vi val, comp, z; int time = 0;
int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
        low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back(); z.pop_back();
        comp[x] = low;
        if (values[x>>1] == -1)
            values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
}

bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
}
};
```

780b64, 15 lines

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:**  $\mathcal{O}(V + E)$

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
```

```
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
        D[x]--, D[y]++;
        eu[e] = 1; s.push_back(y);
    }
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
    return {ret.rbegin(), ret.rend()};
}
```

## 7.5 Coloring

### EdgeColoring.h

**Description:** Given a simple, undirected graph with max degree  $D$ , computes a  $(D + 1)$ -coloring of the edges such that no neighboring edges share a color. ( $D$ -coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

**Time:**  $\mathcal{O}(NM)$

```
vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    rep(i, 0, sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
    return ret;
}
```

## 7.6 Heuristics

### MaximalCliques.h

**Description:** Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.

**Time:**  $\mathcal{O}\left(3^{n/3}\right)$ , much faster for sparse graphs

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
    if (!P.any()) { if (!X.any()) f(R); return; }
    auto q = (P | X)._Find_first();
    auto cands = P & ~eds[q];
    rep(i, 0, sz(eds)) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    }
}
```

### MaximumClique.h

**Description:** Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

**Time:** Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

```
typedef vector<bitset<200>> vb;
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
        rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
    }
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (sz(R)) {
            if (sz(q) + R.back().d <= sz(qmax)) return;
            q.push_back(R.back().i);
            vv T;
            for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
            if (sz(T)) {
                if (S[lev]++ / ++pk < limit) init(T);
                int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
                C[1].clear(), C[2].clear();
                for (auto v : T) {
                    int k = 1;
                    auto f = [&](int i) { return e[v.i][i]; };
                    while (any_of(all(C[k]), f)) k++;
                    if (k > mxk) mxk = k, C[mxk + 1].clear();
                    if (k < mnk) T[j++].i = v.i;
                    C[k].push_back(v.i);
                }
                if (j > 0) T[j - 1].d = 0;
                rep(k, mnk, mxk + 1) for (int i : C[k])
                    T[j].i = i, T[j++].d = k;
                expand(T, lev + 1);
            } else if (sz(q) > sz(qmax)) qmax = q;
            q.pop_back(), R.pop_back();
        }
    }
    vi maxClique() { init(V), expand(V); return qmax; }
    Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
        rep(i, 0, sz(e)) V.push_back({i});
    }
};
```

### MaximumIndependentSet.h

**Description:** To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertex-Cover.

## 7.7 Trees

### 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"0f62fb, 21 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)];}
};
```

### CompressTree.h

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

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

"LCA.h"9775a0, 21 lines

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

### HLD.h

**Description:** Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most  $\log(n)$  light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS\_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.



**Time:**  $\mathcal{O}((\log N)^2)$

```

"../data-structures/LazySegmentTree.h" 03139d, 46 lines
template <bool VALS_EDGES> struct HLD {
    int N, tim = 0;
    vector<vi> adj;
    vi par, siz, rt, pos;
    Node *tree;
    HLD(vector<vi> adj_)
        : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
          rt(N), pos(N), tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
    void dfsSz(int v) {
        if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
        for (int& u : adj[v]) {
            par[u] = v;
            dfsSz(u);
            siz[v] += siz[u];
            if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
        }
    }
    void dfsHld(int v) {
        pos[v] = tim++;
        for (int u : adj[v]) {
            rt[u] = (u == adj[v][0] ? rt[v] : u);
            dfsHld(u);
        }
    }
    template <class B> void process(int u, int v, B op) {
        for (; rt[u] != rt[v]; v = par[rt[v]]) {
            if (pos[rt[u]] > pos[rt[v]]) swap(u, v);
            op(pos[rt[v]], pos[v] + 1);
        }
        if (pos[u] > pos[v]) swap(u, v);
        op(pos[u] + VALS_EDGES, pos[v] + 1);
    }
    void modifyPath(int u, int v, int val) {
        process(u, v, [&](int l, int r) { tree->add(l, r, val); });
    }
    int queryPath(int u, int v) { // Modify depending on problem
        int res = -1e9;
        process(u, v, [&](int l, int r) {
            res = max(res, tree->query(l, r));
        });
        return res;
    }
    int querySubtree(int v) { // modifySubtree is similar
        return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);
    }
};

```

## LinkCutTree.h

**Description:** Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

**Time:** All operations take amortized  $\mathcal{O}(\log N)$ .

```

0fb462, 90 lines
struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
};

```

```

}
int up() { return p ? p->c[1] == this : -1; }
void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
        x->c[h] = y->c[h ^ 1];
        y->c[h ^ 1] = x;
    }
    z->c[i ^ 1] = this;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
}
void splay() {
    for (pushFlip(); p; ) {
        if (p->p) p->p->pushFlip();
        p->pushFlip(); pushFlip();
        int c1 = up(), c2 = p->up();
        if (c2 == -1) p->rot(c1, 2);
        else p->p->rot(c2, c1 != c2);
    }
}
Node* first() {
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
}
};

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

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v));
        makeRoot(&node[u]);
        node[u].pp = &node[v];
    }

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

    bool connected(int u, int v) { // are u, v in the same tree?
        Node* nu = access(&node[u])->first();
        return nu == access(&node[v])->first();
    }

    void makeRoot(Node* u) {
        access(u);
        u->splay();
        if (u->c[0]) {
            u->c[0]->p = 0;
            u->c[0]->flip ^= 1;
            u->c[0]->pp = u;
            u->c[0] = 0;
            u->fix();
        }
    }

    Node* access(Node* u) {
        u->splay();
        while (Node* pp = u->pp) {
            pp->splay(); u->pp = 0;
        }
    }
};

```

```

    if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
    pp->c[1] = u; pp->fix(); u = pp;
}
return u;
}
};

```

## DirectedMST.h

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

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

```

"../data-structures/UnionFindRollback.h" 39e620, 60 lines
struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node(e));
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    rep(s, 0, n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) {
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cycs.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
    }

    for (auto& [u, t, comp] : cycs) { // restore sol (optional)
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
};

```

```
// Given two points, this calculates the smallest angle between
```

```
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

## 8.2 Circles

### CircleIntersection.h

**Description:** Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h"	84d6d3, 11 lines
<pre>typedef Point&lt;double&gt; P; bool circleInter(P a,P b,double r1,double r2,pair&lt;P, P&gt;* out) {     if (a == b) { assert(r1 != r2); return false; }     P vec = b - a;     double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;     if (sum*sum &lt; d2    dif*dif &gt; d2) return false;     P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);     *out = {mid + per, mid - per};     return true; }</pre>	

### CircleTangents.h

**Description:** Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h"	b0153d, 13 lines
<pre>template&lt;class P&gt; vector&lt;pair&lt;P, P&gt;&gt; tangents(P c1, double r1, P c2, double r2) {     P d = c2 - c1;     double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;     if (d2 == 0    h2 &lt; 0) return {};     vector&lt;pair&lt;P, P&gt;&gt; out;     for (double sign : {-1, 1}) {         P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;         out.push_back({c1 + v * r1, c2 + v * r2});     }     if (h2 == 0) out.pop_back();     return out; }</pre>	

### CirclePolygonIntersection.h

**Description:** Returns the area of the intersection of a circle with a ccw polygon.  
**Time:**  $\mathcal{O}(n)$

".../content/geometry/Point.h"	alee63, 19 lines
<pre>typedef Point&lt;double&gt; P; #define arg(p, q) atan2(p.cross(q), p.dot(q)) double circlePoly(P c, double r, vector&lt;P&gt; ps) {     auto tri = [&amp;](P p, P q) {         auto r2 = r * r / 2;         P d = q - p;</pre>	

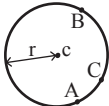
<pre>auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2(); auto det = a * a - b; if (det &lt;= 0) return arg(p, q) * r2; auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det)); if (t &lt; 0    1 &lt;= s) return arg(p, q) * r2; P u = p + d * s, v = p + d * t; return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2; }; auto sum = 0.0; rep(i,0,sz(ps))     sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c); return sum; }</pre>	
--	--

### circumcircle.h

**Description:**

The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

"Point.h"	1caa3a, 9 lines
<pre>typedef Point&lt;double&gt; P; double ccRadius(const P&amp; A, const P&amp; B, const P&amp; C) {     return (B-A).dist()*(C-B).dist()*(A-C).dist()/         abs((B-A).cross(C-A))/2; } P ccCenter(const P&amp; A, const P&amp; B, const P&amp; C) {     P b = C-A, c = B-A;     return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2; }</pre>	



### MinimumEnclosingCircle.h

**Description:** Computes the minimum circle that encloses a set of points.  
**Time:** expected  $\mathcal{O}(n)$

"circumcircle.h"	09dd0a, 17 lines
<pre>pair&lt;P, double&gt; mec(vector&lt;P&gt; ps) {     shuffle(all(ps), mt19937(time(0)));     P o = ps[0];     double r = 0, EPS = 1 + 1e-8;     rep(i,0,sz(ps)) if ((o - ps[i]).dist() &gt; r * EPS) {         o = ps[i], r = 0;         rep(j,0,i) if ((o - ps[j]).dist() &gt; r * EPS) {             o = (ps[i] + ps[j]) / 2;             r = (o - ps[i]).dist();             rep(k,0,j) if ((o - ps[k]).dist() &gt; r * EPS) {                 o = ccCenter(ps[i], ps[j], ps[k]);                 r = (o - ps[i]).dist();             }         }     }     return {o, r}; }</pre>	

## 8.3 Polygons

### InsidePolygon.h

**Description:** Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.  
**Usage:** vector<P> v = {P{4,4}, P{1,2}, P{2,1}};  
bool in = inPolygon(v, P{3, 3}, false);  
**Time:**  $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h"	2bf504, 11 lines
<pre>template&lt;class P&gt; bool inPolygon(vector&lt;P&gt; &amp;p, P a, bool strict = true) {     int cnt = 0, n = sz(p);     rep(i,0,n) {         P q = p[(i + 1) % n];</pre>	

<pre>if (onSegment(p[i], q, a)) return !strict; //or: if (segDist(p[i], q, a) &lt;= eps) return !strict; cnt ^= ((a.y&lt;p[i].y) - (a.y&lt;q.y)) * a.cross(p[i], q) &gt; 0; } return cnt; }</pre>	
---	--

### PolygonArea.h

**Description:** Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h"	f12300, 6 lines
<pre>template&lt;class T&gt; T polygonArea2(vector&lt;Point&lt;T&gt;&gt;&amp; v) {     T a = v.back().cross(v[0]);     rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);     return a; }</pre>	

### PolygonCenter.h

**Description:** Returns the center of mass for a polygon.  
**Time:**  $\mathcal{O}(n)$

"Point.h"	9706dc, 9 lines
<pre>typedef Point&lt;double&gt; P; P polygonCenter(const vector&lt;P&gt;&amp; v) {     P res(0, 0); double A = 0;     for (int i = 0, j = sz(v) - 1; i &lt; sz(v); j = i++) {         res = res + (v[i] + v[j]) * v[j].cross(v[i]);         A += v[j].cross(v[i]);     }     return res / A / 3; }</pre>	

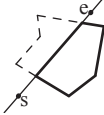
### PolygonCut.h

**Description:**

Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

**Usage:** vector<P> p = ...;  
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "LineIntersection.h"	f2b7d4, 13 lines
<pre>typedef Point&lt;double&gt; P; vector&lt;P&gt; polygonCut(const vector&lt;P&gt;&amp; poly, P s, P e) {     vector&lt;P&gt; res;     rep(i,0,sz(poly)) {         P cur = poly[i], prev = i ? poly[i-1] : poly.back();         bool side = s.cross(e, cur) &lt; 0;         if (side != (s.cross(e, prev) &lt; 0))             res.push_back(lineInter(s, e, cur, prev).second);         if (side)             res.push_back(cur);     }     return res; }</pre>	



### ConvexHull.h

**Description:**

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

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

"Point.h"	310954, 13 lines
<pre>typedef Point&lt;ll&gt; P; vector&lt;P&gt; convexHull(vector&lt;P&gt; pts) {     if (sz(pts) &lt;= 1) return pts;     sort(all(pts));     vector&lt;P&gt; h(sz(pts)+1);     int s = 0, t = 0;     for (int it = 2; it--; s = --t, reverse(all(pts)))</pre>	



```
    for (P p : pts) {
        while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
        h[t++] = p;
    }
    return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

### HullDiameter.h

**Description:** Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

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

"Point.h"	c571b8, 12 lines
<pre>typedef Point&lt;ll&gt; P; array&lt;P, 2&gt; hullDiameter(vector&lt;P&gt; S) {     int n = sz(S), j = n &lt; 2 ? 0 : 1;     pair&lt;ll, array&lt;P, 2&gt;&gt; res({0, {S[0], S[0]}});     rep(i,0,j)         for (; j = (j + 1) % n) {             res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});             if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) &gt;= 0)                 break;         }     return res.second; }</pre>	

### PointInsideHull.h

**Description:** Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

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

"Point.h", "sideOf.h", "OnSegment.h"	71446b, 14 lines
<pre>typedef Point&lt;ll&gt; P;  bool inHull(const vector&lt;P&gt;&amp; l, P p, bool strict = true) {     int a = 1, b = sz(l) - 1, r = !strict;     if (sz(l) &lt; 3) return r &amp;&amp; onSegment(l[0], l.back(), p);     if (sideOf(l[0], l[a], l[b]) &gt; 0) swap(a, b);     if (sideOf(l[0], l[a], p) &gt;= r    sideOf(l[0], l[b], p) &lt;= -r)         return false;     while (abs(a - b) &gt; 1) {         int c = (a + b) / 2;         (sideOf(l[0], l[c], p) &gt; 0 ? b : a) = c;     }     return sgn(l[a].cross(l[b], p)) &lt; r; }</pre>	

### LineHullIntersection.h

**Description:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon:  $\bullet(-1, -1)$  if no collision,  $\bullet(i, -1)$  if touching the corner  $i$ ,  $\bullet(i, i)$  if along side  $(i, i + 1)$ ,  $\bullet(i, j)$  if crossing sides  $(i, i + 1)$  and  $(j, j + 1)$ . In the last case, if a corner  $i$  is crossed, this is treated as happening on side  $(i, i + 1)$ . The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.

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

"Point.h"	7cf45b, 39 lines
<pre>#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n])) #define extr(i) cmp(i + 1, i) &gt;= 0 &amp;&amp; cmp(i, i - 1 + n) &lt; 0 template &lt;class P&gt; int extrVertex(vector&lt;P&gt;&amp; poly, P dir) {     int n = sz(poly), lo = 0, hi = n;     if (extr(0)) return 0;     while (lo + 1 &lt; hi) {         int m = (lo + hi) / 2;         if (extr(m)) return m;         int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);         (ls &lt; ms    (ls == ms &amp;&amp; ls == cmp(lo, m)) ? hi : lo) = m;     }</pre>	

<pre>    }     return lo; }  #define cmpL(i) sgn(a.cross(poly[i], b)) template &lt;class P&gt; array&lt;int, 2&gt; lineHull(P a, P b, vector&lt;P&gt;&amp; poly) {     int endA = extrVertex(poly, (a - b).perp());     int endB = extrVertex(poly, (b - a).perp());     if (cmpL(endA) &lt; 0    cmpL(endB) &gt; 0)         return {-1, -1};     array&lt;int, 2&gt; res;     rep(i,0,2) {         int lo = endB, hi = endA, n = sz(poly);         while ((lo + 1) % n != hi) {             int m = ((lo + hi + (lo &lt; hi ? 0 : n)) / 2) % n;             (cmpL(m) == cmpL(endB) ? lo : hi) = m;         }         res[i] = (lo + !cmpL(hi)) % n;         swap(endA, endB);     }     if (res[0] == res[1]) return {res[0], -1};     if (!cmpL(res[0]) &amp;&amp; !cmpL(res[1]))         switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {             case 0: return {res[0], res[0]};             case 2: return {res[1], res[1]};         }     return res; }</pre>	
--	--

## 8.4 Misc. Point Set Problems

### ClosestPair.h

**Description:** Finds the closest pair of points.

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

"Point.h"	ac41a6, 17 lines
<pre>typedef Point&lt;ll&gt; P; pair&lt;P, P&gt; closest(vector&lt;P&gt; v) {     assert(sz(v) &gt; 1);     set&lt;P&gt; S;     sort(all(v), [](P a, P b) { return a.y &lt; b.y; });     pair&lt;ll, pair&lt;P, P&gt;&gt; ret{LLONG_MAX, {P(), P()}};     int j = 0;     for (P p : v) {         P d{1 + (ll)sqrt(ret.first), 0};         while (v[j].y &lt;= p.y - d.x) S.erase(v[j++]);         auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);         for (; lo != hi; ++lo)             ret = min(ret, {(lo - p).dist2(), {lo, p}});         S.insert(p);     }     return ret.second; }</pre>	

## 8.5 3D

### PolyhedronVolume.h

**Description:** Magic formula for the volume of a polyhedron. Faces should point outwards.

3058c3, 6 lines
<pre>template&lt;class V, class L&gt; double signedPolyVolume(const V&amp; p, const L&amp; trilst) {     double v = 0;     for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]);     return v / 6; }</pre>

### Point3D.h

**Description:** Class to handle points in 3D space. T can be e.g. double or long long.

8058ae, 32 lines
<pre>template&lt;class T&gt; struct Point3D {     typedef Point3D P;     typedef const P&amp; R;     T x, y, z;     explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}     bool operator&lt;(R p) const {         return tie(x, y, z) &lt; tie(p.x, p.y, p.z); }     bool operator==(R p) const {         return tie(x, y, z) == tie(p.x, p.y, p.z); }     P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }     P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }     P operator*(T d) const { return P(x*d, y*d, z*d); }     P operator/(T d) const { return P(x/d, y/d, z/d); }     T dot(R p) const { return x*p.x + y*p.y + z*p.z; }     P cross(R p) const {         return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);     }     T dist2() const { return x*x + y*y + z*z; }     double dist() const { return sqrt((double)dist2()); }     //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]     double phi() const { return atan2(y, x); }     //Zenith angle (latitude) to the z-axis in interval [0, pi]     double theta() const { return atan2(sqrt(x*x+y*y),z); }     P unit() const { return *this/(T)dist(); } //makes dist()==1     //returns unit vector normal to *this and p     P normal(P p) const { return cross(p).unit(); }     //returns point rotated 'angle' radians ccw around axis     P rotate(double angle, P axis) const {         double s = sin(angle), c = cos(angle); P u = axis.unit();         return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;     } };</pre>

### 3dHull.h

**Description:** Computes all faces of the 3-dimension hull of a point set. \*No four points must be coplanar\*, or else random results will be returned. All faces will point outwards.

**Time:**  $\mathcal{O}(n^2)$

5b45fc, 49 lines
<pre>typedef Point3D&lt;double&gt; P3;  struct PR {     void ins(int x) { (a == -1 ? a : b) = x; }     void rem(int x) { (a == x ? a : b) = -1; }     int cnt() { return (a != -1) + (b != -1); }     int a, b; };  struct F { P3 q; int a, b, c; };  vector&lt;F&gt; hull3d(const vector&lt;P3&gt;&amp; A) {     assert(sz(A) &gt;= 4);     vector&lt;vector&lt;PR&gt;&gt; E(sz(A), vector&lt;PR&gt;(sz(A), {-1, -1}));     #define E(x,y) E[f.x][f.y]     vector&lt;F&gt; FS;     auto mf = [&amp;](int i, int j, int k, int l) {         P3 q = (A[j] - A[i]).cross((A[k] - A[i]));         if (q.dot(A[l]) &gt; q.dot(A[i]))             q = q * -1;         F f{q, i, j, k};         E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);         FS.push_back(f);     };     rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)         mf(i, j, k, 6 - i - j - k);</pre>

```
rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
        F f = FS[j];
        if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
            E(a,b).rem(f.c);
            E(a,c).rem(f.b);
            E(b,c).rem(f.a);
            swap(FS[j--], FS.back());
            FS.pop_back();
        }
        int nw = sz(FS);
        rep(j,0,nw) {
            F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
            C(a, b, c); C(a, c, b); C(b, c, a);
        }
        for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
            A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
        return FS;
    };
};
```

sphericalDistance.h

**Description:** Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 ( $\phi_1$ ) and f2 ( $\phi_2$ ) from x axis and zenith angles (latitude) t1 ( $\theta_1$ ) and t2 ( $\theta_2$ ) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx\*radius is then the difference between the two points in the x direction and d\*radius is the total distance between the points.

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

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:  $\mathcal{O}(n)$ 
d4375c, 16 lines
vi pi(const string& s) {
    vi p(sz(s));
    rep(i,1,sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}
```

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

Zfunc.h

**Description:** z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

```
Time:  $\mathcal{O}(n)$ 
ee09e2, 12 lines
vi Z(const string& S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - 1]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}
```

Manacher.h

**Description:** For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).

```
Time:  $\mathcal{O}(N)$ 
e7ad79, 13 lines
array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi,2> p = {vi(n+1), vi(n)};
    rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
        int t = r-i+!z;
        if (i<r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=l && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}
```

MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.

**Usage:** rotate(v.begin(), v.begin()+minRotation(v), v.end());

```
Time:  $\mathcal{O}(N)$ 
d07a42, 8 lines
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) {a = b; break;}
    }
    return a;
}
```

SuffixArray.h

**Description:** Builds suffix array for a string. sa[i] is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size n + 1, and sa[0] = n. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. The input string must not contain any zero bytes.

```
Time:  $\mathcal{O}(n \log n)$ 
769289, 23 lines
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s), y(n), ws(max(n, lim)), rank(n);
        x.push_back(0), sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
        }
    }
};
```

```
rep(i,0,n) ws[x[i]]++;
rep(i,1,lim) ws[i] += ws[i - 1];
for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
swap(x, y), p = 1, x[sa[0]] = 0;
rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
    (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
}
rep(i,1,n) rank[sa[i]] = i;
for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1];
        s[i + k] == s[j + k]; k++);
}
};
```

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:  $\mathcal{O}(26N)$ 
aae0b8, 50 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] = 1[l] = -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;
}
};
```

AhoCorasick.h

Description: Ask Rafael, he must know that shit

a41621, 53 lines

```
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].output = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

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

edce47, 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:  $\mathcal{O}(N \log N)$

9e9d8d, 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;
}
```

10.2 Misc. algorithms

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

b20ecc, 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  $f(b, c) \leq f(a, d)$  and  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a \leq b \leq c \leq d$ . Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

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

DivideAndConquerDP.h

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

Time:  $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 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 Optimization tricks

\_\_builtin\_ia32\_ldmxcsr(40896); disables denormals (which make floats 20x slower near their minimum value).

10.4.1 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).
- $c = x \& -x, r = x + c; ((r \wedge x) >> 2) / c \mid r$  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.

10.4.2 Pragas

- **#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  $[0, 2b)$ .  
751a02, 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;
    }
};
```

FastInput.h  
**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.  
**Usage:** `./a.out < input.txt`  
**Time:** About 5x as fast as `cin/scanf`.  
7b3c70, 17 lines

```
inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin);
    }
    return buf[bc++]; // returns 0 on EOF
}

int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 48;
    return a - 48;
}
```

# Techniques (A)

techniques.txt	159 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Scheduling	
Max contiguous subvector sum	
Invariants	
Huffman encoding	
Graph theory	
Dynamic graphs (extra book-keeping)	
Breadth first search	
Depth first search	
* Normal trees / DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Konig's theorem and vertex cover	
Min-cost max flow	
Lovasz toggle	
Matrix tree theorem	
Maximal matching, general graphs	
Hopcroft-Karp	
Hall's marriage theorem	
Graphical sequences	
Floyd-Warshall	
Euler cycles	
Flow networks	
* Augmenting paths	
* Edmonds-Karp	
Bipartite matching	
Min. path cover	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Edge coloring	
* Trees	
Vertex coloring	
* Bipartite graphs (=> trees)	
* 3^n (special case of set cover)	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Coin change	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
Dynprog over intervals	
Dynprog over subsets	
Dynprog over probabilities	
Dynprog over trees	
3^n set cover	
Divide and conquer	
Knuth optimization	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Bitonic cycle	
Log partitioning (loop over most restricted)	
Combinatorics	

Computation of binomial coefficients
Pigeon-hole principle
Inclusion/exclusion
Catalan number
Pick's theorem
Number theory
Integer parts
Divisibility
Euclidean algorithm
Modular arithmetic
* Modular multiplication
* Modular inverses
* Modular exponentiation by squaring
Chinese remainder theorem
Fermat's little theorem
Euler's theorem
Phi function
Frobenius number
Quadratic reciprocity
Pollard-Rho
Miller-Rabin
Hensel lifting
Vieta root jumping
Game theory
Combinatorial games
Game trees
Mini-max
Nim
Games on graphs
Games on graphs with loops
Grundy numbers
Bipartite games without repetition
General games without repetition
Alpha-beta pruning
Probability theory
Optimization
Binary search
Ternary search
Unimodality and convex functions
Binary search on derivative
Numerical methods
Numeric integration
Newton's method
Root-finding with binary/ternary search
Golden section search
Matrices
Gaussian elimination
Exponentiation by squaring
Sorting
Radix sort
Geometry
Coordinates and vectors
* Cross product
* Scalar product
Convex hull
Polygon cut
Closest pair
Coordinate-compression
Quadtrees
KD-trees
All segment-segment intersection
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Discrete second derivatives
Strings
Longest common substring
Palindrome subsequences

Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Suffix array
Suffix tree
Aho-Corasick
Manacher's algorithm
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Best-first (A*)
Bidirectional search
Iterative deepening DFS / A*
Data structures
LCA (2^k-jumps in trees in general)
Pull/push-technique on trees
Heavy-light decomposition
Centroid decomposition
Lazy propagation
Self-balancing trees
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks
Persistent segment tree