OwO

Cipher La Tasreq

Ezzeldin, Ramez, Mosayed

WELL

Contest (1)

template.cpp

36 lines

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define vi vector<int>
#define all(v) v.begin(), v.end()

void solve() {

}

signed main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int t = 1;
  cin >> t;
  while (t--)
    solve();
  return 0;
}

/* rare stuff:
#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define sz(x) (int)(x).size()
#define uint unsigned long long
typedef pair<int, int> pii;

priority_queue<int, vector<int>, greater<int>> pq;
freopen("file.in", "r", stdin);

mt19937 rng = mt19937(random_device()());

int rand_int(int a, int b) {
    return uniform_int_distribution<int>(a, b)(rng);
}
cin.exceptions(cin.failbit);

*/
```

.vimrc

7 lines

```
set cin ar aw ai is ts=4 sw=4 tm=50 nu noeb ru cul sm
syn on | filetype plugin indent on | colo desert
let &makeprg = 'g++ -Wall -Wconversion -Wfatal-errors -g -std=c
     ++17 -fsanitize=undefined,address % -o %<'

map <F5> :w<CR>:make<CR>:!./%< < %<.in ;read && clear <CR><F13>
au BufNewFile *.cpp 0r ./template.cpp
```

stress.sh

10 lines

```
g++ -o A A.cpp
g++ -o B B.cpp
g++ -o gen gen.cpp
for ((i = 1; ; ++i)); do  # if they are same then will loop
     forever
    echo $i
    ./gen $i > int
    ./A < int > out1
    ./B < int > out2
     diff -w <(./A < int) <(./B < int) || break
done
```

troubleshoot.txt

72 lines

```
General:
Write down most of your thoughts, even if you're not sure
     whether they're useful.
Give your variables (and files) meaningful names
Stay organized and don't leave papers all over the place!
You should know what your code is doing..

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

Wrong answer:
Print your solution! Print debug output as well.
Read the full problem statement again
Make sure your input is correct / same as problem.
Have you understood the problem correctly?
Are you sure your algorithm works?
Try writing a slow (but correct) solution
Can your algorithm handle the whole range of input?
Did you consider corner cases (e.g., n=1)?
Is your output format correct? (including whitespace)
Are you clearing all data structures between test cases?
Any uninitialized variables?
Any undefined behavior (array out of bounds)?
Any overflows or NaNs (or shifting long long by >=64 bits)?
Confusing N and M, i and j, etc.?
Confusing ++i and i++?
Return vs continue vs break?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some test cases 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.
Rewrite your solution from the start or let a teammate do it

Geometry:
Work with ints if possible
Correctly account for numbers close to (but not) zero
 - For functions like acos, make sure the absolute value of the
      input is not (slightly) greater than one.
Correctly deal with vertices that are collinear, concyclic,
     coplanar (in 3D), etc.
Subtracting a point from every other (but not itself)?

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's your complexity? Large TL does not mean that something
     simple (like NlogN) isn't intended.
Are you copying a lot of unnecessary data? (Use references)
Avoid vector, map (Use arrays/unordered_map)
How big is the input and output? (Consider FastIO)
What do your teammates think about your algorithm?
Calling count() on multiset?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
If using pointers, try BumpAllocator
```

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 $x = -b/2a$.

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

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

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

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

2.2 Recurrences

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

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

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

2.3 Trigonometry

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

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$(V+W)\tan(v-w)/2 = (V-W)\tan(v+w)/2$$

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

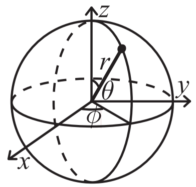
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Sums

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

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.6 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

2.7 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.7.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n, p)$, $n = 1, 2, \ldots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \ \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n, p)$ is approximately $\mathrm{Po}(np)$ for small p.

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1}, \ k = 1, 2, \ldots$$

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

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0, 1, 2, \ldots$$

$$\mu = \lambda, \ \sigma^2 = \lambda$$

2.7.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

Data structures (3)

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.
Time: $\mathcal{O}(\log N)$. Usage: Node* tr = new Node(v, 0,sz(v));

"../various/BumpAllocator.h" 807f30, 77 lines

```
const int inf = 1e9;

struct Node {
  Node *l = 0, *r = 0;
  int lo, hi;
  int mx = -inf, mn = inf, sum = 0;
  int la = 1, lb = 0;

  Node(int lo, int hi) : lo(lo), hi(hi) {}

  Node(vector<int> &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);
      mx = max(l->mx, r->mx);
      mn = min(l->mn, r->mn);
      sum = l->sum + r->sum;
    } else {
      mx = mn = sum = v[lo];
    }
  }

  void push() {
    if (!l) {
      int mid = lo + (hi - lo) / 2;
      l = new Node(lo, mid);
      r = new Node(mid, hi);
    }
    if (la != 1 || lb != 0) {
      l->apply(la, lb);
      r->apply(la, lb);
      la = 1;
      lb = 0;
    }
  }

  void apply(int a, int b) {
    int t1 = mx * a + b;
    int t2 = mn * a + b;

    mx = max(t1, t2);
    mn = min(t2, t1);
    sum = sum * a + b * (hi - lo);

    la = la * a;
    lb = lb * a + b;
  }

  void update(int L, int R, int a, int b) {
    if (R <= lo || hi <= L)
      return;
    if (L <= lo && hi <= R) {
      apply(a, b);
    } else {
      push();
      l->update(L, R, a, b);
      r->update(L, R, a, b);
      mx = max(l->mx, r->mx);
      mn = min(l->mn, r->mn);
      sum = l->sum + r->sum;
    }
  }

  int query(int L, int R) {
    if (R <= lo || hi <= L)
      return -inf;
    if (L <= lo && hi <= R)
      return mx;
    push();
    return max(l->query(L, R), r->query(L, R));
  }

  void set(int L, int R, int x) { update(L, R, 0, x); }
  void add(int L, int R, int x) { update(L, R, 1, x); }
  void mult(int L, int R, int x) { update(L, R, x, 0); }
};
```

MergeSortTree.h

Description: Merge-Sort Tree for Range Queries. The tree stores sorted segments of the array to allow efficient binary search for range queries.
Time: - Construction: $\mathcal{O}(NlogN)$ - Query: $\mathcal{O}(log^2 N)$

093e2e, 32 lines

```
struct MSTree {
  int n;
  vector<vector<int>> s;

  MSTree(vector<int> &a) {
    n = a.size();
    s.resize(2 * n);
    for (int i = 0; i < n; i++)
      s[i + n] = {a[i]};
    for (int i = n - 1; i > 0; i--) {
      auto &L = s[2 * i], &R = s[2 * i + 1];
      auto &P = s[i];
      P.reserve(L.size() + R.size());
      merge(all(L), all(R), back_inserter(P));
    }
  }
  // count of elements > x in [l..r)
  int query(int l, int r, int x) {
    int cnt = 0;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) {
        cnt += s[l].end() - upper_bound(all(s[l]), x);
        l++;
      }
      if (r & 1) {
        --r;
        cnt += s[r].end() - upper_bound(all(s[r]), x);
      }
    }
    return cnt;
  }
};
```

FenwickPURQ.cpp

Description: Point Update, Range Query
Time: $\mathcal{O}(logN)$

52a33a, 32 lines

```
struct FenwickPURQ {
    int n;
    vi f;

    void add(int idx, int val) {
        for (; idx <= n; idx += idx & -idx) f[idx] += val;
    }

    int prefix(int idx) {
        int res = 0;
        for (; idx > 0; idx -= idx & -idx) res += f[idx];
        return res;
    }

    FenwickPURQ(int size) : n(size), f(n + 1, 0) {}

    int rangeQuery(int l, int r) {
        return prefix(r) - prefix(l - 1);
    }

    int lower_bound(int v){
        int sum = 0, pos = 0;
        for(int i = ceil(log2(n)); i >= 0; i--){
            int nextPos = pos + (1 << i);
            if(pos + (1 << i) < n && sum + f[nextPos] < v){
                sum += f[nextPos];
                pos = nextPos;
            }
        }
        return pos + 1;
    }
};
```

FenwickRUPQ.cpp

Description: Range Update, Point Query
Time: $\mathcal{O}(logN)$

478999, 21 lines

```
struct FenwickRUPQ {
    int n;
```

```
    vi f;
    FenwickRUPQ(int _n) : n(_n), f(n + 1, 0) {}

    void update(int idx, int val) {
        for (; idx <= n; idx += idx & -idx)
            f[idx] += val;
    }

    void rangeAdd(int l, int r, int val) {
        update(l, val);
        if (r + 1 <= n) update(r + 1, -val);
    }

    int pointQuery(int idx) {
        int res = 0;
        for (; idx > 0; idx -= idx & -idx) res += f[idx];
        return res;
    }
};
```

FenwickRURQ.cpp

Description: Range Update, Range Query
Time: $\mathcal{O}(logN)$

9c527b, 32 lines

```
struct FenwickRURQ {
    int n;
    vi B1, B2;
    FenwickRURQ(int size) : n(size), B1(n+1, 0), B2(n+1, 0) {}

    void add(vi& f, int idx, int val) {
        for (; idx <= n; idx += idx & -idx) f[idx] += val;
    }

    int prefix(vi& f, int idx){
        int res = 0;
        for (; idx > 0; idx -= idx & -idx) res += f[idx];
        return res;
    }

    void rangeUpdate(int l, int r, int val) {
        add(B1, l, val);
        add(B1, r + 1, -val);
        add(B2, l, val * (l - 1));
        add(B2, r + 1, -val * r);
    }

    int prefixQuery(int idx){
        int sumB1 = prefix(B1, idx);
        int sumB2 = prefix(B2, idx);
        return sumB1 * idx - sumB2;
    }

    int rangeQuery(int l, int r){
        return prefixQuery(r) - prefixQuery(l - 1);
    }
};
```

Fenwick2d.h

Description: Computes sums a[i,j] for all i<N, j<M, and increases single elements a[i,j].
Time: $\mathcal{O}(\log N \cdot \log M)$.

e4e159, 24 lines

```
struct Fenwick2D {
    int n, m;
    vector<vi> f;

    Fenwick2D(int _n, int _m) : n(_n), m(_m), f(n + 1, vi(m +
         1, 0)) {}

    void update(int x, int y, int val) {
        for (int i = x; i <= n; i += i & -i)
            for (int j = y; j <= m; j += j & -j)
                f[i][j] += val;
    }

    int prefixSum(int x, int y) const {
        int res = 0;
        for (int i = x; i > 0; i -= i & -i)
            for (int j = y; j > 0; j -= j & -j)
                res += f[i][j];
        return res;
    }

    int rangeSum(int x1, int y1, int x2, int y2) const {
        return prefixSum(x2, y2) - prefixSum(x1 - 1, y2) -
            prefixSum(x2, y1 - 1) + prefixSum(x1 - 1, y1 - 1);
    }
};
```

Fenwick2dAdd.h

Description: Handles RURQ for sums
Time: $\mathcal{O}(\log N \cdot \log M)$.

4e7d75, 51 lines

```
struct Fenwick2DAdd {
    int n, m;
    vector<vi> T1, T2, T3, T4;

    Fenwick2DAdd(int _n, int _m)
        : n(_n), m(_m),
          T1(n+1, vi(m+1)),
          T2(n+1, vi(m+1)),
          T3(n+1, vi(m+1)),
          T4(n+1, vi(m+1))
    {}

    void add(vector<vi>& t, int x, int y, int v) {
        for (int i = x; i <= n; i += i & -i)
            for (int j = y; j <= m; j += j & -j)
                t[i][j] += v;
    }

    void rangeAdd(int x, int y, int v) {
        add(T1, x, y, v);
        add(T2, x, y, v * (y - 1));
        add(T3, x, y, v * (x - 1));
        add(T4, x, y, v * (x - 1) * (y - 1));
    }

    void rangeUpdate(int x1, int y1, int x2, int y2, int val) {
        rangeAdd(x1, y1, val);
        rangeAdd(x1, y2 + 1, -val);
        rangeAdd(x2 + 1, y1, -val);
        rangeAdd(x2 + 1, y2 + 1, val);
    }

    int prefixSum(int x, int y) const {
        int s1 = 0, s2 = 0, s3 = 0, s4 = 0;
        for (int i = x; i > 0; i -= i & -i)
            for (int j = y; j > 0; j -= j & -j) {
                s1 += T1[i][j];
                s2 += T2[i][j];
                s3 += T3[i][j];
                s4 += T4[i][j];
            }
        return s1 * x * y - s2 * x - s3 * y + s4;
    }

    int rangeQuery(int x1, int y1, int x2, int y2) const {
        return prefixSum(x2, y2)
            - prefixSum(x1 - 1, y2)
            - prefixSum(x2, y1 - 1)
            + prefixSum(x1 - 1, y1 - 1);
    }
};
```

Fenwick2dXor.h

Description: Handles RURQ for XOR
Time: $\mathcal{O}(\log N \cdot \log M)$.

a2ef3c, 44 lines

```
struct Fenwick2DXOR {
    int n, m;
    vector<vi> bit[2][2];

    Fenwick2DXOR(int _n, int _m) : n(_n), m(_m) {
        for (int px = 0; px < 2; ++px)
            for (int py = 0; py < 2; ++py)
                bit[px][py].assign(n+2, vi(m+2, 0));
    }

    void pointXOR(int x, int y, int v) {
        for (int i = x; i <= n; i += i & -i)
            for (int j = y; j <= m; j += j & -j)
                bit[x&1][y&1][i][j] ^= v;
    }

    void rangeXOR(int x1, int y1, int x2, int y2, int v) {
        if (x1 > x2 || y1 > y2) return;
        auto upd = [&](int x, int y){ if (x > 0 && y > 0)
            pointXOR(x, y, v); };
        upd(x1,     y1);
        upd(x2 + 1, y1);
        upd(x1,     y2 + 1);
        upd(x2 + 1, y2 + 1);
    }

    int prefixXOR(int x, int y) {
        if (x <= 0 || y <= 0) return 0;
        int res = 0;
        int px = x & 1, py = y & 1;
        for (int i = x; i > 0; i -= i & -i)
            for (int j = y; j > 0; j -= j & -j)
                res ^= bit[px][py][i][j];
        return res;
    }

    int rangeQuery(int x1, int y1, int x2, int y2) {
        int res = 0;
        res ^= prefixXOR(x2,   y2);
        res ^= prefixXOR(x1-1, y2);
        res ^= prefixXOR(x2,   y1-1);
        res ^= prefixXOR(x1-1, y1-1);
        return res;
    }
};
```

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).

a925b7, 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 = (int)(4e18 * acos(0)) | 71;
  int operator()(int x) const { return __builtin_bswap64(x*C);
      }
};
__gnu_pbds::gp_hash_table<int,int,chash> h({},{},{},{},{1<<16})
    ;
```

DSU.h

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

1fba21, 21 lines

```
struct DSU {
    vector<int> parent, size;
    int count; // of component

    DSU(int n) : parent(n + 1), size(n + 1, 1), count(n) { iota
        (all(parent), 0); }

    int find(int i) { return (parent[i] == i ? i : (parent[i] =
        find(parent[i]))); }

    bool same(int i, int j) { return find(i) == find(j); }

    int getSize(int i) { return size[find(i)]; }

    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1;
        else --count;
        if (size[i] > size[j]) swap(i, j);
        parent[i] = j;
        size[j] += size[i];
        return j;
    }
};
```

DSURollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: `int t = uf.time(); ...; uf.rollback(t);`
Time: $\mathcal{O}(\log(N))$

de4ad0, 21 lines

```
struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}(N^2 + Q)$

c59ada, 13 lines

```
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

Matrix.h

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

6ccb3b, 26 lines

```
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;
  }
  array<T, N> operator*(const array<T, N>& vec) const {
    array<T, N> ret{};
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(int 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: $\mathcal{O}(\log N)$

b518fa, 30 lines

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

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const int inf = LLONG_MAX;
  int div(int a, int 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(int k, int 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));
  }
  int query(int 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: $\mathcal{O}(\log N)$

1754b4, 53 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 [L,R] = split(n->l, k);
    n->l = R;
    n->recalc();
    return {L, n};
  } else {
    auto [L,R] = split(n->r,k - cnt(n->l) - 1); // and just "k"
    n->r = L;
    n->recalc();
    return {n, R};
  }
}
```

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

Node* ins(Node* t, Node* n, int pos) {
  auto [l,r] = split(t, pos);
  return merge(merge(l, n), r);
}

// 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));
}
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V|\log|V| + Q)$ 510c32, 16 lines

```
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)]);
  }
};
```

MoQueries.h

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change `step` to add/remove the edge (a, c) and remove the initial `add` call (but keep `in`).
Time: $\mathcal{O}(N\sqrt{Q})$ a12ef4, 49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vi mo(vector<pii> Q) {
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) {
    pii q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}

vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
  int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) -> void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) rep(end,0,2) {
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                  else { add(c, end); in[c] = 1; } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
  return res;
}
```

MergeSortTree.h

Description: Merge-Sort Tree for Range Queries. The tree stores sorted segments of the array to allow efficient binary search for range queries.
Time: - Construction: $\mathcal{O}(NlogN)$ - Query: $\mathcal{O}(log^2 N)$ 093e2e, 32 lines

```
struct MSTree {
  int n;
  vector<vector<int>> s;

  MSTree(vector<int> &a) {
    n = a.size();
    s.resize(2 * n);
    for (int i = 0; i < n; i++)
      s[i + n] = {a[i]};
    for (int i = n - 1; i > 0; i--) {
      auto &L = s[2 * i], &R = s[2 * i + 1];
      auto &P = s[i];
      P.reserve(L.size() + R.size());
      merge(all(L), all(R), back_inserter(P));
    }
  }
  // count of elements > x in [l..r)
  int query(int l, int r, int x) {
    int cnt = 0;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) {
        cnt += s[l].end() - upper_bound(all(s[l]), x);
        l++;
      }
      if (r & 1) {
        --r;
        cnt += s[r].end() - upper_bound(all(s[r]), x);
      }
    }
    return cnt;
  }
};
```

BinaryTrie.cpp

Description: binary trie that supports update(val, op), where op = +1 to insert, op = −1 to erase and query(x) → ans is the maximum XOR you can achieve between x and any value currently in the trie.
Time: $\mathcal{O}(1)$ time, $\mathcal{O}(N)$ space 2e58b0, 48 lines

```
struct node {
    int ch[2]{}, frq[2]{}, sz{};

    int& operator[](int x) {
        return ch[x];
    }
};

const int M = 60;

struct BinaryTrie {
    vector<node> nodes;

    int newNode() { return nodes.emplace_back(), nodes.size() -
          1; }

    void init() { nodes.clear(), newNode(); }

    BinaryTrie() { init(); }

    void update(int val, int op) {
        int u = 0;
        for (int i = M - 1; i >= 0; --i) {
            int v = val >> i & 1;
            if (!nodes[u][v]) {
                nodes[u][v] = newNode();
            }
            nodes[u].frq[v] += op;
            nodes[u].sz += op;
            u = nodes[u][v];
        }
        nodes[u].sz++;
    }

    int query(int x) {
        int ans = 0, u = 0;
        for (int i = M - 1; i >= 0 && u >= 0; --i) {
            int v = x >> i & 1;
            if (nodes[u].frq[v]) {
                u = nodes[u][v];
            }
            else {
                u = nodes[u][!v];
                ans |= 1LL << i;
            }
        }
        return ans;
    }
};
```

SQRTD.cpp

Description: Square-root decomposition for range sum queries with point updates. Preprocesses the array into blocks of size sqrt(n), maintaining the sum of each block. query(l, r): returns the sum of arr[l..r] in O(sqrt(n)) time. update(idx, val): updates arr[idx] to val and adjusts the corresponding block sum in O(1). When to use: - You have an array of size n (up to 10^5) with mixed range-sum queries and point updates. - You need better performance than O(n) per operation but segment trees are overkill.
Time: $\mathcal{O}(sqrt(n))$ per query, $\mathcal{O}(1)$ per update.

<bits/stdc++.h> f564e5, 71 lines

```
using namespace std;

using vi = vector<int>;

int n, q, SQ;
vi arr, blkSum;

// Build block sums from the initial array in O(n)
void preProcess() {
    // For each element, add it to its block's sum
    for (int i = 0; i < n; ++i) {
        int blk = i / SQ;
        blkSum[blk] += arr[i];
    }
}

// Query the sum in the interval [l, r] in O(sqrt(n))
int query(int l, int r) {
    int ans = 0;
    // Process elements until we reach block boundary or exceed r
    while (l <= r) {
        // If l is at the start of a block and the whole block lies within [l, r]
        if (l % SQ == 0 && l + SQ - 1 <= r) {
            ans += blkSum[l / SQ];
            l += SQ;
        } else {
            // Otherwise, take the single element
            ans += arr[l];
            ++l;
        }
    }
    return ans;
}

// Point update: set arr[idx] = val in O(1)
void update(int idx, int val) {
    int blk = idx / SQ;
    // Remove old value from block sum and add new value
    blkSum[blk] -= arr[idx];
    arr[idx] = val;
    blkSum[blk] += val;
}

void solve() {
    // Read array size and number of operations
    cin >> n >> q;
    arr.resize(n);
    cin >> arr;

    // Determine block size and initialize block sums
    SQ = ceil(sqrt(n));
    blkSum.assign((n + SQ - 1) / SQ, 0);

    preProcess();

    while (q--) {
        int op;
        cin >> op;
        if (op == 1) {
            // Update operation: 1 pos val
            int pos, val;
            cin >> pos >> val;
            update(pos - 1, val);
        } else {
            // Query operation: 2 l r
            int l, r;
            cin >> l >> r;
            cout << query(l - 1, r - 1) << "\n";
        }
    }
}
```

MOs.cpp

Description: Mo's algorithm (offline) for answering range-distinct queries. Sorts queries into sqrt-blocks by L, then R, and moves two pointers to maintain current range, updating a frequency table.
Time: $\mathcal{O}((N+Q)*sqrt(N))$ Memory $\mathcal{O}(N+Q)$. 87503f, 62 lines

```
struct Query {
    int l, r, idx;
    bool operator<(const Query &other) const {
        const int BLOCK = 450;             // sqrt(max(N))
        int b1 = l / BLOCK;
        int b2 = other.l / BLOCK;
        if (b1 != b2) return b1 < b2;      // different block by L
        return r < other.r;                // same block, sort by R
    }
};

void solve() {
    fastio();
    int n, q;
    cin >> n >> q;
    vector<int> a(n);
    for (int &x : a) cin >> x;

    // Read and store offline queries
    vector<Query> queries(q);
    for (int i = 0; i < q; ++i) {
        int l, r;
        cin >> l >> r;
        queries[i] = {l - 1, r - 1, i};    // convert to 0-based
    }

    // Sort queries by Mo's ordering
    sort(queries.begin(), queries.end());

    // Frequency table, current distinct count, current range [curL..curR]
    const int MAXV = 1'000'005;
    vector<int> freq(MAXV, 0);
    int distinct = 0, curL = 0, curR = -1;
    vector<int> ans(q);

    // Process each query by expanding/shrinking [curL..curR]
    for (auto &qr : queries) {
        // Expand right end
        while (curR < qr.r) {
            if (++freq[a[++curR]] == 1) ++distinct;
        }
        // Shrink right end
        while (curR > qr.r) {
            if (--freq[a[curR--]] == 0) --distinct;
        }
        // Shrink left end
        while (curL < qr.l) {
            if (--freq[a[curL++]] == 0) --distinct;
        }
        // Expand left end
        while (curL > qr.l) {
            if (++freq[a[--curL]] == 1) ++distinct;
        }
        // Record answer for this query
        ans[qr.idx] = distinct;
    }

    // Output all answers in original order
    for (int x : ans) {
        cout << x << '\n';
    }
}
```

Numerical (4)

4.1 Fourier transforms

FastFourierTransform.h

Description: 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: `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 $N = |A| + |B|$ (~1s for $N = 2^{22}$) 00ced6, 35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (static int k = 2; k < 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&1 ? R[i/2] * x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      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& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

FastFourierTransformMod.h

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

"FastFourierTransform.h" 8dc350, 22 lines

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

NumberTheoreticTransform.h

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

"../number-theory/ModPow.h" 72dcaf, 56 lines

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

// int generator () {
//     vector<int> fact;
//     int phi = mod-1,   n = phi;
//     for (int i=2; i*i<=n; ++i)
//         if (n % i == 0) {
//             fact.push_back (i);
//             while (n % i == 0)
//                 n /= i;
//         }
//     if (n > 1)
//         fact.push_back (n);
//
//     for (int res=2; res<=mod; ++res) {
//         bool ok = true;
//         for (size_t i=0; i<fact.size() && ok; ++i)
//             ok &= modpow (res, phi / fact[i]) != 1;
//         if (ok)  return res;
//     }
//     return -1;
// }
```

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

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

"euclid.h" dfe297, 18 lines

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

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime.

33cd6e, 3 lines

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

ModPow.h

50496a, 8 lines

```
const int mod = 1000000007; // faster if const

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

ModLog.h

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

9b1195, 11 lines

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

ModSum.h

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

3be38d, 16 lines

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

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

int modsum(uint to, int c, int k, int 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`

1ca7e1, 11 lines

```
typedef unsigned long long uint;
uint modmul(uint a, uint b, uint M) {
  int ret = a * b - M * uint(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (int)M);
}
uint modpow(uint b, uint e, uint mod) {
  uint 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" 50a30e, 24 lines

```
int sqrt(int a, int 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
  int 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;
  int x = modpow(a, (s + 1) / 2, p);
  int b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    int t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    int gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

IsPrime.cpp

Description: Checks if a number is prime or not
Time: $\mathcal{O}\left(\sqrt{n}\right)$

8c0c7a, 8 lines

```
bool isPrime(int n) {
    if (n == 2) return true;
    if (n == 1 || n % 2 == 0) return false;
    for (int i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return false;
    }
    return true;
}
```

Sieve.h

Description: Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.
Time: lim=$10^8 \approx 0.8$ s. Runs 30% faster if only odd indices are stored.

6dfb4f, 14 lines

```
const int MAX_PR = 5'000'000;
bitset<MAX_PR> isprime;
vi sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;

  for (int i = 2; i < lim; i++)
        if (isprime[i]) pr.push_back(i);

  return pr;
}
```

FastSieve.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;
}
```

LinearSieve.cpp

Description: just like normal sieve but faster
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

a18245, 14 lines

```
vi linearSieve(int n) {
  vector<bool> isPr(n + 1, 1);
  vi primes;
  isPr[0] = isPr[1] = 0;
  for (int i = 2; i <= n; i++) {
    if (isPr[i]) primes.push_back(i);
    for (int p : primes) {
      if (i * p >= n + 1) break;
      isPr[i * p] = 0;
      if (i % p == 0) break;
    }
  }
  return primes;
}
```

SieveSpf.h

Description: Computes the smallest prime factor (SPF) for every number up to N using a sieve. Can be used for fast prime factorization in $O(\log n)$ per query after $O(N \log\log N)$ preprocessing.
Time: sieve - $\mathcal{O}(N \log\log N)$, factorization - $\mathcal{O}(\log n)$

6b1488, 14 lines

```
int NMAX = 1e6;
vi spf(NMAX + 1, 1);

spf[0] = 0; spf[1] = 1;
for (int i = 2; i <= NMAX; ++i)
    if (spf[i] == 1)
        for (int j = i; j <= NMAX; j += i)
            if (spf[j] == 1)
                spf[j] = i;

while (x > 1) {
    primes[spf[x]]++;
    x /= spf[x];
}
```

SieveDivs.h

Description: Computes all divisors for every number in the range $[1, n)$. Returns a vector of vectors where result[i] contains all divisors of i.
Time: $\mathcal{O}(n \log n)$

4e7a53, 5 lines

```
vector<vi> divs(1e6);
for (int i = 1; i < sz(divs); ++i)
    for (int j = i; j < sz(divs); j += i)
        divs[j].push_back(i);
    }
```

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 $a^b \mod c$.

"ModMulLL.h" fd7b28, 12 lines

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

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h" dc6e12, 18 lines

```
uint pollard(uint n) {
  uint x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](uint 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<uint> factor(uint n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  uint x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

PrimeFactorization.cpp

Description: gets the prime factors of a number
Time: $\mathcal{O}\left(\sqrt{n}\right)$

e01cc9, 12 lines

```
vector<pair<int, int>> getPrimeFactors(int n) {
    vector<pair<int, int>> primeFactors;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            int count = 0;
            while (n % i == 0) n /= i, count++;
            primeFactors.push_back({i, count});
        }
    }
    if (n > 1) primeFactors.push_back({n, 1});
    return primeFactors;
}
```

5.3 Divisibility

euclid.h

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

1ea825, 5 lines

```
int euclid(int a, int b, int &x, int &y) {
  if (!b) return x = 1, y = 0, a;
```

```
  int d = euclid(b, a % b, y, x);
  return y -= a/b * x, d;
}
```

CRT.h

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

"euclid.h" 190cc6, 7 lines

```
int crt(int a, int m, int b, int n) {
  if (n > m) swap(a, b), swap(m, n);
  int x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

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

cf7d6d, 8 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

5.4 Combinatorics

Combinatorics.cpp

Description: Function to solve combinatorics problems.
Time: $\mathcal{O}(n)$ for init and $\mathcal{O}(1)$ for query

260fa1, 56 lines

```
namespace combinatorics {
    vector<int> fact, inv, invFact;

    int pwmod(int a, int b) {
        a %= MOD;
        int result = 1;
        while (b > 0) {
            if (b & 1) result = (result * a) % MOD;
            a = (a * a) % MOD;
            b /= 2;
        }
        return result;
    }

    int inverse(int x) { return pwmod(x, MOD - 2); }
    int multiply(int a, int b) { return ((a % MOD) * (b % MOD))
        % MOD; }
    int divide(int a, int b) { return multiply(a, inverse(b));
    }

    void init(int n) {
        fact.resize(n + 1); inv.resize(n + 1); invFact.resize(n
            + 1);
        fact[0] = fact[1] = inv[0] = inv[1] = invFact[0] =
            invFact[1] = 1;
        for (int i = 2; i <= n; ++i){
            fact[i] = fact[i - 1] * i % MOD;
            inv[i] = MOD - ((MOD / i) * inv[MOD % i]) % MOD;
            invFact[i] = invFact[i - 1] * inv[i] % MOD;
        }
    }

    int nPr(int n, int r) {
        if (n < 0 || r < 0 || r > n) return 0;
        return fact[n] * invFact[n - r] % MOD;
    }

    int nCr(int n, int r) {
        if (n < 0 || r < 0 || r > n) return 0;
        return fact[n] * invFact[r] % MOD * invFact[n - r] %
            MOD;
    }

    int nPrLinear(int n, int r){
        int answer = 1;
        for (int i = n - r + 1; i <= n; i++){
            answer = multiply(answer, i);
        }
        return answer;
    }

    int nCrLinear(int n, int r){
        int answer = 1;
        for (int i = r + 1; i <= n; i++){
            answer = multiply(answer, i);
            answer = divide(answer, i - r);
        }
        return answer;
    }
};
using namespace combinatorics;
```

Ncr.h

Description: Precomputes factorials and inverse factorials modulo mod, call build_fact once before using nCr.

"ModPow.h" dbdebb, 20 lines

```
vector<int> fact = {1}, inv = {1};

void build_fact(int n = 2e6) {

  inv.resize(n + 1);

  for (int i = 1; i <= n; i++)
    fact.push_back(fact.back() * i % mod);

  inv[n] = modpow(fact[n], mod - 2);
  for (int i = n - 1; i >= 0; --i)
    inv[i] = inv[i + 1] * (i + 1) % mod;
}
int ncr(int n, int r) {
  if (r < 0 || r > n)
    return 0;
  return fact[n] * inv[r] % mod * inv[n - r] % mod;
  // For npr: return fact[n] * inv[n - r] % mod;
  // ncr(n + r - 1, n) for stars and bars
}
```

Ncr2.h

ef0363, 12 lines

```
int nCr(int n, int r) {
  // Useful when r is very small
  if (r > n)
    return 0;
  r = min(r, n - r);
  int ans = 1;
  for (int i = 0; i < r; i++) {
    ans = ans * (n - i) % mod;
    ans = ans * modpow(i + 1, mod - 2) % mod;
  }
  return ans;
}
```

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 Estimates

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

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

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 Derangements

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Binomials

multinomial.h

Description: Computes $\binom{k_1+\dots+k_n}{k_1,k_2,\dots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!\dots k_n!}$. fdf149, 5 lines

```
int multinomial(vi& v) {
  int 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). $B[0,\dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Catalan numbers

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

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

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

- 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

dfs.cpp

Description: Traversing graph
Time: $\mathcal{O}(V+E)$ fef44b, 16 lines

```
int n;
vector<vi> adj;
vi vis(n + 1), ans;

void dfs(int u) {
    vis[u] = true;
    cout << u << ' ';

    for (int v : graph[u]) {
        if (!vis[v]) {
            dfs(v);
        }
    }

    ans.push_back(u);
}
```

bfs.cpp

Description: Traversing graph
Time: $\mathcal{O}(V+E)$ b27965, 21 lines

```
vector<vi> adj;
vi vis(n + 1, 0), p(n + 1, -1);

void bfs(int start) {
    queue<int> q; q.push(start);
    p[start] = -1;

    while (!q.empty()) {
        int u = q.front(); q.pop();
        cout << u << endl;
        vis[u] = 1;
        for (int v : adj[u]) {
            if (!vis[v]) {
                vis[v] = 1;
                q.push(v);
                p[v] = g;
                cout << v << endl;
            }
        }
    }
}
```

7.2 Paths

dijkstra.cpp

Description: Find shortest path from node 1 to all nodes.
Time: V = 1e5, E = 1e6. a3be53, 27 lines

```
int n, m; cin >> n >> m;
vector<vector<pii>> adj(n + 1);

for (int i = 0; i < m; i++) {
  int a, b, c; cin >> a >> b >> c;
  adj[a].push_back({ b, c });
}

vi vis(n + 1), dis(n + 1);
priority_queue<pii, vector<pii>, greater<pii>> pq; // {cost,
    node}
pq.push({ 0, 1 });
```

```
while (!pq.empty()) {
  auto [parentCost, u] = pq.top(); pq.pop();
  if (vis[u]) continue;
  vis[u] = 1; dis[u] = parentCost;

  for (auto [v, childCost] : adj[u]) {
    if (!vis[v]) {
      pq.push({ parentCost + childCost, v });
    }
  }
}

for (int i = 1; i <= n; i++) {
  cout << dis[i] << " ";
}
```

dijkstraK.cpp

Description: Find the k shortest routes from 1 to n
Time: $\mathcal{O}(E + V)$ 7b3b58, 23 lines

```
int n, m, k; cin >> n >> m >> k;
vector<vector<pii>> adj(n + 1);

for (int i = 0; i < m; i++) {
    int a, b, c; cin >> a >> b >> c;
    adj[a].push_back({ b, c });
}

vector<vi> dis(n + 1);
priority_queue<pii, vector<pii>, greater<pii>> pq; // {cost,
     node}
pq.push({ 0, 1 });

while (!pq.empty()) {
    auto [parentCost, u] = pq.top(); pq.pop();
    if (dis[u].size() >= k) continue;
    dis[u].push_back(parentCost);

    for (auto [v, childCost] : adj[u]) {
        pq.push({ parentCost + childCost, v });
    }
}

cout << dis[n] << "\n";
```

BellmanFord.h

Description: Find maximum score to travel from 1 to n, negative allowed, infinite cycles allowed
Time: V = 500 a6376d, 55 lines

```
struct edge {
    int u, v, w;
};

int n, m; cin >> n >> m;

vector<edge> edges;

for (int i = 0; i < m; i++) {
    int u, v, w; cin >> u >> v >> w;
    edges.push_back({ u, v, w });
}

vi score(n + 1, LLONG_MIN);
score[1] = 0;

// Relaxation (n - 1) times
for (int i = 1; i <= n - 1; i++) {
    for (int j = 0; j < m; j++) {
        auto [u, v, w] = edges[j];
        if (score[u] != LLONG_MIN) {
            score[v] = max(score[v], score[u] + w);
        }
    }
}

/*
After the initial relaxation steps, we check if any edge can
      still be relaxed.
If it can, that means there's a cycle (specifically a "positive
       cycle" for maximizing the score)
that can improve the score.
*/
vector<bool> hasPositiveCycle(n + 1, false);
for (int i = 0; i < m; i++) {
    auto [u, v, w] = edges[i];
    if (score[u] != LLONG_MIN && score[v] < score[u] + w) {
        hasPositiveCycle[v] = true;
    }
}

/*
However, simply detecting an edge that can be relaxed doesn't
      tell us which vertices
might be affected downstream by this cycle.
The propagation loop iterates over all edges several times (in
      this case, n times)
to "spread" the effect of the positive cycle
*/
for (int i = 1; i <= n; i++) {
    for (int j = 0; j < m; j++) {
        auto [u, v, w] = edges[j];
        if (hasPositiveCycle[u]) hasPositiveCycle[v] = true;
    }
}

if (hasPositiveCycle[n]) cout << -1 << "\n";
else cout << score[n] << "\n";
```

floyedWarshall.cpp

Description: Find shortest path from all nodes to all nodes
Time: V = 5000, E = 1e6 50e2d0, 27 lines

```
int n, m, q; cin >> n >> m >> q;

vector<vi> dis(n + 1, vi(n + 1, LLONG_MAX));
for (int i = 1; i <= n; i++) {
    dis[i][i] = 0;
}

for (int i = 0; i < m; i++) {
    int a, b, c; cin >> a >> b >> c;
    dis[a][b] = min(dis[a][b], c);
    dis[b][a] = min(dis[b][a], c);
}

for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (dis[i][k] < LLONG_MAX && dis[k][j] < LLONG_MAX)
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j
                    ]);
        }
    }
}

while (q--) {
    int a, b; cin >> a >> b;
    if (dis[a][b] == LLONG_MAX) cout << "-1\n";
    else cout << dis[a][b] << "\n";
}
```

TopologicalSort.cpp

Description: A topological sort takes a directed acyclic graph (DAG) and produces, a linear ordering of its vertices such that for every directed edge u -> v, u comes before v in that order, Returns a vector of nodes in a valid order; if a cycle exists, the size will be < n.
Time: $\mathcal{O}(E + V)$ 279399, 20 lines

```
vi topologicalSort(int n, vector<vi>& adj, vi& inDeg) {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        if (inDeg[i] == 0)
            q.push(i);
    }

    vi order;

    while (!q.empty()) {
        int u = q.front(); q.pop();
        order.push_back(u);
        for (int v : adj[u]) {
            if (--inDeg[v] == 0)
                q.push(v);
        }
    }

    return order;
}
```

DAGLongestPathDP.cpp

Description: What is the maximum number of cities I can visit on any directed path from 1 to n in a graph with no cycles DAG?
Time: $\mathcal{O}(V + E)$ 40cc1c, 21 lines

```
vi order = topologicalSort(n, adj, inDeg);

vi dp(n + 1, -1), parent(n + 1, -1);
dp[1] = 1;

for (int u : order) {
    if (dp[u] < 0)    // not reachable from 1
        continue;

    for (int v : adj[u]) {
        if (dp[u] + 1 > dp[v]) {
            dp[v] = dp[u] + 1;
            parent[v] = u;
        }
    }
}

if (dp[n] < 0) {
    cout << "IMPOSSIBLE\n";
    return;
}
```

7.3 Cycles

CountCyclesDFS.cpp

Description: Counts cycles in graph, if this function returned true, count++.
Time: $\mathcal{O}(V + E)$ fc9d66, 14 lines

```
bool countCyclesDFS(int u) {
    visited[u] = true;

    for (int v : graph[u]) {
```

```
        if (!visited[v]) {
            if (countCyclesDFS(v)) {
                return true;
            }
        } else if (v != u) {
            return true;
        }
    }
    return false;
}
```

findingACycleInGraph.cpp

Description: Finds a path for cycle in graph
Time: $\mathcal{O}(V + E)$

53d278, 36 lines

```
// Color: 0 = unvisited, 1 = in-stack, 2 = done
vi color(n + 1, 0), parents(n + 1, -1), cycle;
bool found = false;

function<bool(int)> dfs = [&](int u) -> bool {
    color[u] = 1;
    for (int v : adj[u]) {
        if (color[v] == 0) {
            parents[v] = u;
            if (dfs(v)) return true;
        }
        else if (color[v] == 1) {
            // back edge u -> v found a cycle
            found = true;
            cycle.push_back(v);
            for (int x = u; x != v; x = parents[x])
                cycle.push_back(x);
            cycle.push_back(v);
            reverse(all(cycle));
            return true;
        }
    }
    color[u] = 2;
    return false;
};

for (int i = 1; i <= n && !found; i++) {
    if (color[i] == 0) dfs(i);
}

if (!found) {
    cout << "IMPOSSIBLE\n";
} else {
    cout << cycle.size() << "\n";
    cout << cycle << "\n";
}
```

7.4 Componenets

ConnectedComponenetsBFS.cpp

Description: Count number of connected componenets.
Time: $\mathcal{O}(V + E)$

5956dd, 24 lines

```
int cnt = 0;
vi vis(n + 1, 0);

for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
        q.push(i);
        vis[i] = 1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            vis[u] = 1;
            for (int v : adj[u]) {
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push(v);
                    p[v] = u;
                }
            }
        }
        cnt++;
    }
}

cout << "cnt = " << cnt << endl;
```

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);
}
```

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

76b5c9, 24 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
  int low = val[j] = ++Time, x; z.push_back(j);
  for (auto e : g[j]) if (comp[e] < 0)
    low = min(low, val[e] ?: dfs(e,g,f));

  if (low == val[j]) {
    do {
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
  }
  return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
  int n = sz(g);
  val.assign(n, 0); comp.assign(n, -1);
  Time = ncomps = 0;
  rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

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

780b64, 15 lines

```
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()};
}
```

7.6 Trees

LCAAndKthAncestor.h

Description: Data structure for computing lowest common ancestors in a tree C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q)$

9ff08f, 53 lines

```
struct Tree {
    int n, LOG;
    vi depth;
    vector<vi> up;

    Tree(const vector<vi>& adj, int root = 0) {
        n = adj.size();
        LOG = ceil(log2(n));
        depth.assign(n, 0);
        up.assign(LOG + 1, vi(n, -1));

        dfs(adj, root, root);
```

```
        for (int k = 1; k <= LOG; ++k) {
            for (int v = 0; v < n; ++v) {
                int p = up[k - 1][v];
                up[k][v] = (p < 0 ? -1 : up[k - 1][p]);
            }
        }
    }

    // To get the parent and depth of each node
    void dfs(const vector<vi>& adj, int v, int parent) {
        up[0][v] = parent;
        for (int u : adj[v]) {
            if (u == parent) continue;
            depth[u] = depth[v] + 1;
            dfs(adj, u, v);
        }
    }

    int kth_ancestor(int v, int dist) const {
        for (int k = 0; dist && v >= 0; ++k) {
            if (dist & 1) v = up[k][v];
            dist >>= 1;
        }
        return v;
    }

    int LCA(int a, int b) const {
        if (depth[a] < depth[b]) swap(a, b);
        a = kth_ancestor(a, depth[a] - depth[b]);
        if (a == b) return a;
        for (int k = LOG; k >= 0; --k) {
            if (up[k][a] != up[k][b]) {
                a = up[k][a];
                b = up[k][b];
            }
        }
        return up[0][a];
    }
};
```

KruskalMST.cpp

Description: it finds the minimum cost of forming a minimum spanning tree, if it can't be formed it returns -1, it can also help detecting cycles easily in graph

Time: $\mathcal{O}(mlogm)$

64836f, 19 lines

```
struct Edge {
    int u, v, w;
    Edge() : u(0), v(0), w(0) {}
    Edge(int u, int v, int w) : u(u), v(v), w(w) {}
    bool operator<(Edge const &other) const { return w < other.
        w; }
};

int kruskalMST(vector<Edge> &edges, int n) {
    sort(all(edges));
    int cost = 0; DSU dsu(n + 1);
    for (auto &[u, v, w] : edges) {
        if (!dsu.same(u, v)) {
            cost += w;
            dsu.merge(u, v);
        }
    }
    if (dsu.getSize(dsu.find(1)) == n) return cost;
    return -1;
}
```

7.7 Math

7.7.1 Number of Spanning Trees

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

7.7.2 Erdős–Gallai theorem

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

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

Strings (8)

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 - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

Manacher.h

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

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

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>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
  }
  return p;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

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

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

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 nul chars.

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

635552, 22 lines

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string s, int lim=256) { // or vector<int>
    s.push_back(0); int n = sz(s), k = 0, a, b;
    vi x(all(s)), y(n), ws(max(n, lim));
    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++;
    }
    for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
      for (k && k--, j = sa[x[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] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
  }

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

Hashing.h

Description: Self-explanatory methods for string hashing.

234773, 75 lines

```
constexpr int H = 2;
typedef array<long long, H> val;
vector<val> B;
const val M = {
    1000000007, 1444444447,
    // 998244353,
    // 1000000009,
};

val tmp;

val operator+(const val &a, const val &b) {
  for (int i = 0; i < H; i++)
    tmp[i] = (a[i] + b[i]) % M[i];
  return tmp;
}

val operator*(const val &a, const val &b) {
  for (int i = 0; i < H; i++)
    tmp[i] = a[i] * b[i] % M[i];
  return tmp;
}

val operator-(const val &a, const val &b) {
  for (int i = 0; i < H; i++)
    tmp[i] = (a[i] - b[i] + M[i]) % M[i];
  return tmp;
}

val getval(int x) {
  // make sure x is always positvie if not handle it
  for (int i = 0; i < H; i++)
    tmp[i] = x % M[i];
  return tmp;
}

void setB(int n) {
  if (B.size() == 0) {
    mt19937 rng(random_device{}());
    B.assign(2, getval(1));
    for (int i = 0; i < H; i++)
      B.back()[i] = uniform_int_distribution<int>(1, M[i] - 1)(
          rng);
  }
  while ((int)B.size() <= n)
    B.push_back(B.back() * B[1]);
}

struct Hash {
  vector<val> h;

  Hash(const string &s) : Hash(vector<int>(all(s))) {}

  Hash(const vector<int> &s) {
    vector<val> v;
    for (auto x : s)
      v.push_back(getval(x));
    *this = Hash(v);
  }

  Hash(const vector<val> &s) : h(s.size() + 1) {
    setB(s.size());
    for (int i = 0; i < (int)s.size(); i++)
      h[i + 1] = h[i] * B[1] + s[i];
  }
  val get(int l, int r) { return h[r + 1] - h[l] * B[r - l +
      1]; }
};

// val concat(val &a, val &b, int len_b) { return a * B[len_b]
    + b; }
//
// struct val_hash {
//   size_t operator()(const val &v) const {
//     return hash<int>{}(v[0]) ^ (hash<int>{}(v[1]) << 1);
//   }
// };
```

Trie.h

Description: trie that supports update(val, op), where op = +1 to insert, op = −1 to erase
Time: $\mathcal{O}(1)$ time, $\mathcal{O}(N)$ space

522d36, 42 lines

```
struct Trie {
    struct Node {
        int ch[26]{};
        int cnt = 0, end = 0;
    };
    vector<Node> t = {{}};

    void update(string s, int op) {
        int u = 0;
        for (char c : s) {
            int v = c - 'a';
            if (!t[u].ch[v]) {
                if (op == -1) return;
                t[u].ch[v] = t.size();
                t.push_back({});
            }
            u = t[u].ch[v];
            t[u].cnt += op;
        }
        t[u].end += op;
    }

    bool find(string s) {
        int u = 0;
        for (char c : s) {
            int v = c - 'a';
            if (!t[u].ch[v]) return false;
            u = t[u].ch[v];
        }
        return t[u].end > 0;
    }

    int prefix(string s) {
        int u = 0;
        for (char c : s) {
            int v = c - 'a';
            if (!t[u].ch[v]) return 0;
            u = t[u].ch[v];
        }
        return t[u].cnt;
    }
};
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$.

f35677, 66 lines

```
struct AhoCorasick {
  enum {alpha = 26, first = 'A'}; // change this!
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(next)); }
  };
  vector<Node> N;
  vi backp;
```

```
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c : s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i,0,alpha) {
        int &ed = N[n].next[i], y = N[prev].next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
            = N[y].end;
          N[ed].nmatches += N[y].nmatches;
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // int count = 0;
    for (char c : word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
  }
  vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

Various (9)

9.1 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to $<=$, and reverse the loop at (B). To minimize f, change it to $>$, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$ — 9155b4, 11 lines

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

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$ — 2932a0, 17 lines

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

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$ — b20ccc, 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;
}
```

9.2 Dynamic programming

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: $\mathcal{O}((N + (hi - lo)) \log N)$ — 80cf3a, 18 lines

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

9.3 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

9.4 Optimization tricks

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

9.4.1 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^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `rep(b,0,K) rep(i,0,(1 << K))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];`
  computes all sums of subsets.

9.4.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

FastMod.h

Description: Compute $a\%b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to a (mod b) in the range $[0, 2b)$. — e38900, 8 lines

```
typedef unsigned long long uint;
struct FastMod {
  uint b, m;
```

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

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
```