German University in Cairo

Triple C

Nour Ayman, Ahmed Mamdouh, Karim Elmosalamy

ACM-GUCCPC

August 21, 2021

Contest (1)

template.cpp

25 lines

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define mst(x) memset(x,0,sizeof(x))
#define mcpy(a,b) memcpy(a, b, sizeof(a)) //arr copying into, arr copying from
#define lcase(s) transform(s.begin(), s.end(), s.begin(), ::tolower)
#define ucase(s) transform(s.begin(), s.end(), s.begin(), ::toupper)
#define getInts(nums,n) for(ll p=0;p<n;p++)cin >> nums[p]

int main() {
  ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0);
  freopen("input.txt","r",stdin);
  freopen("output.txt","w",stdout);

  cerr << "Run time: " << fixed << setprecision(3) << (double)clock() / CLOCKS_PER_SEC << "s" << endl;
  return 0;
}
```

troubleshoot.txt

52 lines

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

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

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

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

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

Mathematics (2)

2.1 Equations

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

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

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

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

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

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

2.2 Recurrences

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

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

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

2.3 Trigonometry

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

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

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

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

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

2.4 Geometry

2.4.1 Volumes

Volume of Sphere : $\frac{4}{3} * \pi * r * r * r$
Surface area of Sphere : $4 * \pi * r * r$
Volume of Cone : $\pi * r * r * \frac{h}{3}$
Surface area of Cone : $\pi * r * (r + \sqrt{(h * h) + (r * r)})$
Volume of Cylinder: $\pi * r * r * h$
Surface area of Cylinder : $2 * \pi * r * h + 2 * \pi * r * r$
Volume of Cuboid : $l * w * h$
Surface area of Cuboid : $2 * (l * w) + 2 * (l * h) + 2 * (h * w)$
Volume of square based Pyramid: $\frac{l * w * h}{3}$
Surface area of square based Pyramid $= l * w + l * \sqrt{(w/2) * (w/2) + (h * h)} + w * \sqrt{(l/2) * (l/2) + (h * h)}$

2.4.2 Triangles

Side lengths: a, b, c
Semiperimeter: $p = \frac{a + b + c}{2}$
Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$
Circumradius: $R = \frac{abc}{4A}$
Inradius: $r = \frac{A}{p}$
Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

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

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

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

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

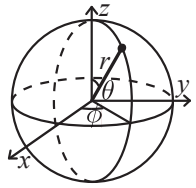
2.4.3 Quadrilaterals

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 $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.4.4 Spherical coordinates



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

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

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

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

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

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

2.7 Series

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

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

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

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

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

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ 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,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

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 $\operatorname{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.9 Markov chains

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

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

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

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

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

Data structures (3)

OrderStatisticTree.h

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

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> 1e8b01, 22 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>;

using namespace __gnu_pbds;
template<class T> using oset=tree<T, null_type, less<T>,
     rb_tree_tag, tree_order_statistics_node_update>;
template<class T,class T2> using omap=tree<T, T2, less<T>,
     rb_tree_tag, tree_order_statistics_node_update>;
template<class T> using omultiset=tree<T, null_type, less_equal
     <T>, rb_tree_tag, tree_order_statistics_node_update>;
template<class T,class T2> using omultimap=tree<T, T2,
     less_equal<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

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

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

SegmentTree.h

Description: One-indexed max-tree. Bounds are inclusive to the left and inclusive to the right.
Time: $\mathcal{O}(\log N)$

f9ac31, 85 lines

```
const int N = 3e5+5;
int ST[N];
int a[N];
int n,q;
int lazy[N];

void buildST(int node=1, int l=1, int r=n){
    if(l == r){
        ST[node]=a[node+1-n];
        return;
    }
    int mid = l + r >> 1;
    int leftChild = node << 1;
    int rightChild = node << 1 | 1;
    buildST(leftChild, l, mid);
    buildST(rightChild, mid+1, r);
    ST[node] = ST[leftChild] + ST[rightChild];
}

void propagate(int node, int l, int r){
    int leftChild = node << 1;
    int rightChild = node << 1 | 1;
    int mid = l + r >> 1;
    ST[leftChild]+=lazy[node]*(mid - l + 1);
    ST[rightChild]+=lazy[node]*(r - mid);
    lazy[leftChild]+=lazy[node];
    lazy[rightChild]+=lazy[node];
    lazy[node]=0;
}

int query(int i, int j, int node=1, int l=1, int r=n){
    if(i <= l && r <= j)
        return ST[node];
    if(r < i || l > j)
        return 0; // neutrlizing agent that doesn't affect the
             answer.
    propagate(node, l, r);
    int mid = l + r >> 1;
    int leftChild = node << 1;
    int rightChild = node << 1 | 1;
    int left = query(i, j, leftChild, l, mid);
    int right = query(i, j, rightChild, mid+1, r);
    return left + right;
}

void update_point(int i, int val){
    int node = i + n - 1;
    ST[node] += val;
    while(node!=1){
        node>>=1;
        int leftChild = node << 1;
        int rightChild = node << 1 | 1;
        ST[node]=ST[leftChild]+ST[rightChild];
    }
}

void updateRange(int i, int j, int val, int node=1, int l=1,
     int r=n){
    if(i <= l && r <= j){
        ST[node]+=val*(r-l+1);
        lazy[node]+=val;
        return;
    }
    if(r < i || l > j)return;
    propagate(node, l, r);
    int mid = l + r >> 1;
    int leftChild = node << 1;
    int rightChild = node << 1 | 1;
    updateRange(i, j, val, leftChild, l, mid);
    updateRange(i, j, val, rightChild, mid+1, r);
    ST[node] = ST[leftChild]+ST[rightChild];
}

int main()
{
    cin >> n;
    for(int i=1; i<=n; i++)cin >> a[i];
    int newSize = 1;
    while(newSize<n)newSize<<=1;
    n=newSize;
    buildST();
    cin >> q;
    while(q--){
        int l,r; cin >> l >> r;
        cout << query(l, r) << '\n';
    }
}
```

LazySegmentTree.h

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

"../various/BumpAllocator.h" 34ecf5, 50 lines

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

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

mergeSortTree.h

Description: Merge Sort Segment Tree

d494af, 43 lines

```
const int maxn = 1e5+10;
class Bit
{
public:
    vector<int> arr ;
    vector<vector<int>> tree ;
    Bit (vector<int> v)
    {
        arr.resize(maxn) ;  tree.resize(4*maxn) ;
        for (int i=0 ; i<v.size() ; i++) arr[i]=v[i] ;
    }
    void build(int node, int l, int r) {
        if(l == r) {
            tree[node].push_back(arr[l]);
            return;
        }
        int mid = (l + r) >> 1,
                left = node << 1, right = left|1;

        build(left, l, mid);
        build(right, mid+1, r);

        merge(all(tree[left]), all(tree[right]),
              back_inserter(tree[node]));
    }
    int query(int node, int l, int r, int i, int j, int k) {
        if(i > r || l > j) return 0;
        if(i <= l && r <= j) {
            return lower_bound(all(tree[node]), k)
                   - tree[node].begin();
        }
        int mid = (l + r) >> 1,
                left = node << 1, right = left|1;
        return query(left, l, mid, i, j, k) +
               query(right, mid+1, r, i, j, k);
    }
};

int32_t main()
{
    Bit tree2 (red2); tree2.build(1,0,red2.size()-1) ;
    ans+=tree2.query(1,0,red2.size()-1,0,i,red2[i]) ;
}
```

UnionFind.h

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

aa17d0, 27 lines

```
const int N = 2e5+5;

int parent[N];
int setSize[N];
int rnk[N];

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

void unionSet(int i, int j){
    i = findSet(i), j = findSet(j);
    if(i == j)return;
    if(rnk[i] > rnk[j]){
        parent[j] = i;
        setSize[i] += setSize[j];
    }else{
        parent[i] = j;
        setSize[j] += setSize[i];
        if(rnk[j] == rnk[i])rnk[j]++;
    }
}

int main()
{
    for(int i=0; i<N; i++)parent[i] = i, setSize[i] = 1, rnk[i]
         = 0;
}
```

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

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}}}};`
`vector<int> vec = {1,2,3};`
`vec = (A^N) * vec;`

c43c7d, 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;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ret(N);
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

LineContainer.h

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

8ec1c7, 30 lines

```
struct Line {
  mutable ll k, m, p;
```

```
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

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

FenwickTree.h

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

e62fac, 22 lines

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

SparseTable.h

Description: Sparse Table
Time: $\mathcal{O}(1)$

e6f390, 26 lines

```
const int N = 2e5+5;
int sparse[N][20], a[N];
int LOG[N];
int n;

int query(int L, int R){
    int length = R - L + 1;
    int k = LOG[length];
    return __gcd(sparse[L][k], sparse[R - (1 << k) + 1][k]);
}

int32_t main()
{
    ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    LOG[1] = 0;
    for(int i=2; i<N; i++)LOG[i] = LOG[i/2] + 1;
    int t; cin >> t;
    while(t--){
        cin >> n;
        for(int i=0; i<n; i++)cin >> a[i];
        for(int i=0; i<n; i++)sparse[i][0] = abs(a[i+1] - a[i])
            ;
        for(int j = 1; j < 20; j++)
            for(int i=0; i + (1 << j) - 1 < n - 1; ++i)
                sparse[i][j] = __gcd(sparse[i][j-1], sparse[i +
                      (1 << (j-1))][j-1]);
    }
}
```

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: $\mathcal{O}\left(N\sqrt{Q}\right)$

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;
}
```

SqrtDecomposition.h

Description: Sqrt Decomposition Code
Time: $\mathcal{O}\left(sqrt(n)\right)$

dcdf04, 26 lines

```
// input data
int n;
vector<int> a (n);

// preprocessing
int len = (int) sqrt (n + .0) + 1; // size of the block and the
      number of blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];

// answering the queries
for (;;) {
    int l, r;
  // read input data for the next query
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to [l,
                r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}
```

Number theory (4)

4.1 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" 35bfea, 18 lines

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

```
};
```

ModInverse.h

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

6f684f, 5 lines

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

// To get inverse of one value, use modular power with power = mod - 2
```

ModPow.h

b83e45, 8 lines

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

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

ModLog.h

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

c040b8, 11 lines

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

ModSum.h

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

5c5bc5, 16 lines

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

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

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

ModMulLL.h

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

bbbd8f, 11 lines

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

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModPow.h" 19a793, 24 lines

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

4.2 Primality

Eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

998f1a, 17 lines

```
const int N = 1e6+5;
int prime[N];
void sieve(){
    for(int i=2; i<N; i++){
        if(prime[i])continue;
        prime[i] = i;
        for(int j=i*i; j<N; j+=i)
            prime[j]=i;
    }
    //getting prime factors of the number!
    for(int i=1; i<N; i++){
        int x = i;
        while(x != 1){
            x /= prime[x];
        }
    }
}
```

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s

6b2912, 20 lines

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

MillerRabin.h

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

"ModMulLL.h" 60dcd1, 12 lines

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

Factor.h

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

"ModMulLL.h", "MillerRabin.h" a33cf6, 18 lines

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

4.3 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 $a \pmod b$.

33ba8f, 5 lines

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

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" 04d93a, 7 lines

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

4.3.1 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;
}
```

4.4 Fractions

ContinuedFractions.h

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

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

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

dd6c5e, 21 lines

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

FracBinarySearch.h

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

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

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

27ab3e, 25 lines

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

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

4.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

4.6 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

4.7 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 $n < 1e19$.

4.8 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

Combinatorial (5)

5.1 Permutations

5.1.1 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;
}
```

5.1.2 Cycles

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

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

5.1.3 Derangements

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

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

5.1.4 Burnside's lemma

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

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

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

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

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

5.2 Partitions and subsets

5.2.1 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

5.2.2 Lucas' Theorem

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

5.2.3 Binomials

bionomialcoeff.h

Description: Gets binomial coefficient 'nCk'. inverse is modular inverse function or precomputed.

2df5a3, 3 lines

```
long long binomial_coefficient(int n, int k) {
    return factorial[n] * inverse(factorial[k] * factorial[n -
        k] % m) % m;
}
```

5.3 General purpose numbers

5.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

5.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

5.3.3 Eulerian numbers

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

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

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

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

5.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

5.3.5 Bell numbers

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

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

5.3.6 Labeled unrooted trees

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

5.3.7 Catalan numbers

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

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

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

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

Graph (6)

6.1 Fundamentals

Dijkstra.h

Description: Tarjan

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

cfe437, 25 lines

```
const int N = 2e5+5, OO = 1e9;
vector<vector<pair<int,int>>> v(N); // {node, cost}
int n;

int dijkstra(int S, int T){
    int dis[N];
    for(int i=0; i<N; i++)dis[i] = OO;
    dis[S] = 0;
    priority_queue<pair<int, int>> pq;
    pq.push({0, S});
    while(pq.size()){
        auto x = pq.top();
        pq.pop();
        x.first*=-1;
        if(x.second == T)return dis[T];
        if(dis[x.second] < x.first)continue;
        for(auto h : v[x.second]){
            if(h.second + x.first < dis[h.first]){
                dis[h.first] = h.second + x.first;
                pq.push({-dis[h.first], h.first});
            }
        }
    }
    return -1;
}
```

BellmanFord.h

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

2ce862, 23 lines

```
bool bellmanFord(int S){ //O(V*E)
    int dis[N];
    for(int i=0; i<N; i++)dis[i] = OO;
    dis[S] = 0;
    bool modified = 1;
    for(int i=0; i<n-1 && modified; i++){
        modified = 0;
        for(int j=0; j<n; j++){
            for(auto h : v[j]){
                if(dis[j] + h.second < dis[h.first]){
                    dis[h.first] = dis[j] + h.second;
                    modified = 1;
                }
            }
        }
    }
    bool hasNegativeCycle = 0;
    for(int i = 0; i<n; i++)
        for(auto h : v[i])
            if(dis[i] + h.second < dis[h.first])
                hasNegativeCycle = 1;
    return hasNegativeCycle;
}
```

FloydWarshall.h

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

6de731, 26 lines

```
const int N = 500, OO = 1e9;

int adjMat[N][N];
int parent[N][N];
int n;

void floyd(){  //O(V^3)
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            if(!adjMat[i][j] && i != j)adjMat[i][j] = OO;
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            parent[i][j] = i;
    for(int k=0; k<n; k++)
        for(int i=0; i<n; i++)
            for(int j=0; j<n; j++)
                if(adjMat[i][j] > adjMat[i][k] + adjMat[k][j]){
                    adjMat[i][j] = adjMat[i][k] + adjMat[k][j];
                    parent[i][j] = parent[k][j];
                }
}

void printPath(int i, int j){
    if(i != j)printPath(i, parent[i][j]);
    cout << j << " ";
}
```

TopoSort.h

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

66a137, 14 lines

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

6.2 Network flow

maxflow.h

Description: Finds maxflow in a graph.
Time: $\mathcal{O}(V * E^2)$

291cf4, 47 lines

```
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

MinCut.h

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

GlobalMinCut.h

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

8b0e19, 21 lines

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

6.3 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. *btoa*[i] will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

f612e4, 42 lines

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a]) if (B[b] == L + 1) {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
  return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    for (int lay = 1;; lay++) {
```

```
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); dfsMatching(g, btoa);`
Time: $\mathcal{O}(VE)$

522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
  if (btoa[j] == -1) return 1;
  vis[j] = 1; int di = btoa[j];
  for (int e : g[di])
    if (!vis[e] && find(e, g, btoa, vis)) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
  vi vis;
  rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h" da4196, 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m);
  for (int it : match) if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    lfound[i] = 1;
    for (int e : g[i]) if (!seen[e] && match[e] != -1) {
      seen[e] = true;
      q.push_back(match[e]);
    }
  }
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

WeightedMatching.h

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

1e0fe9, 31 lines

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

GeneralMatching.h

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

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

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

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

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

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

6.4 DFS algorithms

SCC.h

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

9f75c3, 43 lines

```
const int N = 2e5+5;

vector<vector<int>> v;
int dfs_num[N], inSCC[N], dfs_low[N];
int counter, SCC;
stack<int> st;
int n;

void tarjanSCC(int u){
    dfs_low[u] = dfs_num[u] = ++counter;
    st.push(u);
    for(auto x : v[u]){
        if(!dfs_num[x])
            tarjanSCC(x);
        if(!inSCC[x])
            dfs_low[u] = min(dfs_low[x], dfs_low[u]);
    }
    if(dfs_num[u] == dfs_low[u]){
        SCC++;
        int x = -1;
        do{
            x = st.top(); st.pop();
            inSCC[x]=1;
        }while(u != x);
    }
}

void tarjanSCC(){
    for(int i=0; i<n; i++)
        if(!dfs_num[i])
            tarjanSCC(i);
}

void _clear(){
    v.clear();
    for(int i=0; i<n; i++){
        dfs_low[i]=0;
        dfs_num[i]=0;
```

```
        inSCC[i]=0;
        counter = 0;
        SCC=0;
    }
}
```

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

2965e5, 33 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, e, y, top = me;
  for (auto pa : ed[at]) if (pa.second != par) {
    tie(y, e) = pa;
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me) st.push_back(e);
      else { /* e is a bridge */ }
    }
  }
  return top;
}

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

2sat.h

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

5f9706, 56 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true

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

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

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

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

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

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

flatteningTree.h

Description: Flattens a tree.

7bd2a1, 45 lines

```
int n;
int starting [(int)1e5+5] ;
int ending [(int)1e5+5] ;
vector<vector<int>>adj ;
int FAT[(int)1e5+5] ;
int timer =1 ;
void dfs_flating(int node,int par)
{
    FAT[timer]=node ;
    starting[node]=timer ;
    timer++ ;
    for(auto child:adj[node])
    {
        if(par!=child)
        {
            dfs_flating(child,node) ;

        }

    }
    ending[node]=timer ;
    FAT[timer]=node ;
    timer++ ;
}

int32_t main() {
    iso;
    cin>> n ;
    adj.resize(n+5) ;
    for(int i=0 ;i<n-1 ;i++)
    {
        int u ,v ;
        cin>>u>>v ;
        adj[u].push_back(v) ;
        adj[v].push_back(u) ;
    }
    dfs_flating(1,-1) ;
    /*
    FAT is the new tree , if we wanna to know a subtree of a
         node , we can traverse from starting time of it
    from FAT to ending time of it from FAT
    Starting and ending are start and end time of each node
    array start  index > Node
        */

}
```

6.5 Coloring

EdgeColoring.h

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

e210e2, 31 lines

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

```
      for (int& z = free[y] = 0; adj[y][z] != -1; z++);
  }
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
  return ret;
}
```

6.6 Heuristics

articulationPoints.h

Description: Finding articulation points. Main function is find_cutpoints. it performs necessary initialization and starts depth first search in each connected component of the graph. Function IS_CUTPOINT(a) is some function that will process the fact that vertex aa is an articulation point, for example, print it (Caution that this can be called multiple times for a vertex).
Time: $\mathcal{O}(N+M)$

23f413, 37 lines

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

findingBridges.h

Description: Main function is find_bridges; it performs necessary initialization and starts depth first search in each connected component of the graph. Function 'IS_BRIDGE(a, b)' is some function that will process the fact that edge (a,b)(a,b) is a bridge, for example, print it. Note that this implementation malfunctions if the graph has multiple edges, since it ignores them. Of course, multiple edges will never be a part of the answer, so IS_BRIDGE can check additionally that the reported bridge is not a multiple edge. Alternatively it's possible to pass to dfs the index of the edge used to enter the vertex instead of the parent vertex (and store the indices of all vertices).
Time: $\mathcal{O}(N+M)$

a44485, 33 lines

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

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

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

findingCycles.h

Description: Checking a graph for acyclicity and finding a cycle. Here is an implementation for directed graph.
Time: $\mathcal{O}(M)$

552488, 99 lines

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }
    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v
            ])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}

/*
    Here is an implementation for undirected graph. Note that
        in the undirected version, if a vertex v gets colored
        black, it will never be visited again by the DFS. This
         is because we already explored all connected edges of
         v when we first visited it. The connected component
        containing v (after removing the edge between v and
        its parent) must be a tree, if the DFS has completed
        processing v without finding a cycle. So we don't even
         need to distinguish between gray and black states.
        Thus we can turn the char vector color into a boolean
        vector visited.
*/
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> parent;
int cycle_start, cycle_end;
bool dfs(int v, int par) { // passing vertex and its parent
     vertex
    visited[v] = true;
    for (int u : adj[v]) {
        if(u == par) continue; // skipping edge to parent
             vertex
        if (visited[u]) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
        parent[u] = v;
        if (dfs(u, parent[u]))
            return true;
    }
    return false;
}
void find_cycle() {
    visited.assign(n, false);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (!visited[v] && dfs(v, parent[v]))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
```

```
        for (int v = cycle_end; v != cycle_start; v = parent[v
            ])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

6.7 Trees

LCA.h

Description: Data structure for computing lowest common ancestors in a tree.
Time: $\mathcal{O}(N \log N + Q)$

a274f4, 48 lines

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

CompressTree.h

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

"LCA.h" 9775a0, 21 lines

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

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

"../data-structures/UnionFindRollback.h" 39e620, 60 lines

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

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

  for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  rep(i,0,n) par[i] = in[i].a;
  return {res, par};
}
```

6.8 Math

6.8.1 Number of Spanning Trees

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

6.8.2 Erdős–Gallai theorem

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

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

Geometry (7)

7.1 Geometric primitives

Point.h

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

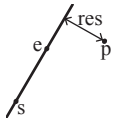
47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
```

```
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

lineDistance.h

Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

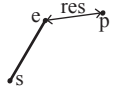


"Point.h" f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h

Description:
Returns the shortest distance between point p and the line segment from point s to e.



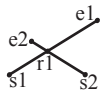
Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```

"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



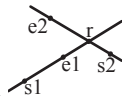
Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```

"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.



Usage:
```
auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
```

"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage:
```
bool left = sideOf(p1,p2,q)==1;
```

"Point.h" 3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

OnSegment.h

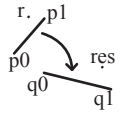
Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h

Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.



"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

LineProjectionReflection.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab insted. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
  P v = b - a;
  return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage:
```
vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively
oriented triangles with vertices at 0 and i
```

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

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

7.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
```

```
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

CircleTangents.h

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

"Point.h" b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

"Point.h" e0cfba, 9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

CirclePolygonIntersection.h

Description: Returns the area of the intersection of a circle with a ccw polygon.

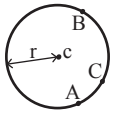
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

circumcircle.h

Description:



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

"Point.h" 1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.

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

"circumcircle.h" 09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

7.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);

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

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

PolygonArea.h

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

"Point.h" f12300, 6 lines

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

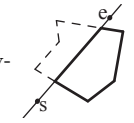
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

PolygonCut.h

Description:



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

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

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

Time: $\mathcal{O}(N^2)$, where N is the total number of points

"Point.h", "sideOf.h" 3931c6, 33 lines

```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
```

```
          segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

7.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

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

7.5 3D

Point3D.h

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

8058ae, 32 lines

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

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: $\mathcal{O}(n)$

f6d312, 13 lines

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

Zfunc.h

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

3ae526, 12 lines

```
vi Z(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 zero bytes.
Time: $\mathcal{O}(n \log n)$

38db9f, 23 lines

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

Hashing.h

Description: Self-explanatory methods for string hashing.

3f02d8, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
struct H {
  typedef uint64_t ull;
  ull x; H(ull x=0) : x(x) {}
#define OP(O,A,B) H operator O(H o) { ull r = x; asm \
  (A "addq %%rdx, %0\n adcq $0,%0" : "+a"(r) : B); return r; }
  OP(+,,"d"(o.x)) OP(*,"mul %1\n", "r"(o.x) : "rdx")
  H operator-(H o) { return *this + ~o.x; }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get() == o.get(); }
  bool operator<(H o) const { return get() < o.get(); }
```

```
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)

struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};

vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

AhoCorasick.h

Description: Aho-Corasick 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; // ll 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 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$ edce47, 23 lines

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

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

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$ 9e9d8d, 19 lines

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

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$ 753a4c, 19 lines

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

9.2 Misc. algorithms

TernarySearch.h

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

Time: $\mathcal{O}(\log(b-a))$

a06956, 26 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;
}

double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x
         ) in [l, r]
}
```

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;
}
```

9.3 Dynamic programming

KnuthDP.h

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

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

DivideAndConquerDP.h

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

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

38e804, 35 lines

```
int m, n;
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid
            ), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int solve() {
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

9.4 Debugging tricks

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

9.5 Optimization tricks

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

9.5.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `rep(b,0,K) rep(i,0,(1 << K))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];`
  computes all sums of subsets.

9.5.2 Pragmas

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

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
```