De Anza College

Team Reference Docs

DA Flint

Sunho Kim, Ralph Cao, Jiyoo Dojun

2023-05-21

Contest (1)

template.cpp — 7 lines

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

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

run.sh — 10 lines

```
basename "$PWD"
g++ -Wall -Wconversion -fsanitize=undefined,address --std=c++17
      -o program *.cpp || exit
files=$(ls *.in)
for file in $files; do
  name="${file%.*}"
  echo "$name running"
  [ -s $file ] || { echo "? skipped "; continue; }
  ./program < "$file" > "/tmp/$name.out" || { echo "!! crashed"
      ; exit; }
  diff -y "/tmp/$name.out" "$name.ans"  || { echo "!! failed (L
      : output | R: answer)"; exit; }
done
```

Mathematics (2)

2.1 Equations

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

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

2.2 Trigonometry

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

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

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

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

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

2.3 Geometry

2.3.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

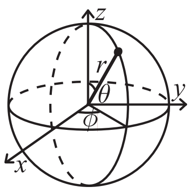
$$s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

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

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

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

2.3.2 Spherical coordinates



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

2.4 Sums

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

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

2.5 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 $\sigma^2=V(X)=\mathbb{E}(X^2)-(\mathbb{E}(X))^2=\sum_x(x-\mathbb{E}(X))^2p_X(x)$ 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,

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

2.5.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

Poisson distribution

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

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

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

2.5.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

2.6 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ 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: $\mathcal{O}(\log N)$

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

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

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

HashMap.h

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

```
#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: Segment tree using binary magic.
Time: $\mathcal{O}(\log N)$

```
template <class T>
struct SegTree {
  const T def = 0; // default value
  int n;
  vector<T> tree;
  SegTree() = default;
  SegTree(int n) : n(n), tree(n * 2, def) {}
  T combine(T a, T b) { return a + b; }
  void init() {
    for (int i = n - 1; i > 0; i--)
      tree[i] = combine(tree[i << 1], tree[i << 1 | 1]);
  }
  void update(int k, T x) {
    k += n, tree[k] = x;
    for (k >>= 1; k; k >>= 1)
      tree[k] = combine(tree[k << 1], tree[k << 1 | 1]);
  }
  // query on [l,r)
  T query(int l, int r) {
    T resl = def, resr = def;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) resl = combine(resl, tree[l++]);
      if (r & 1) resr = combine(tree[--r], resr);
    }
    return combine(resl, resr);
  }
};
using ST = SegTree<long long>;

pair<int, int> interval(ST st, int i) {
  if (i >= st.n)
    return {i - st.n, i - st.n + 1};
  pair<int, int> l = interval(st, i * 2);
  pair<int, int> r = interval(st, i * 2 + 1);
  if (l.second != r.first)
    return {-1, -1};
  return {l.first, r.second};
}

void debug(ST st) {
  for (int i = 1; i < 2 * st.n; i++) {
    auto res = interval(st, i);
    cout << i << ": [" << res.first << ", " << res.second << ")
        " << st.tree[i] << endl;
  }
}
```

LazySegmentTree.h

Description: Lazy segment tree for increment/query on range using binary magic.
Time: $\mathcal{O}(\log N)$.

```
template <class T, class L>
struct LazySegTree {
```

```
  const T def = 0; // change here (remove const to support
       assignment)
  const L ldef = 0; // change here
  int n, h;
  vector<T> tree;
  vector<L> lazy;
  LazySegTree() = default;
  LazySegTree(int n) : n(n), tree(n * 2, def), lazy(n, ldef), h
       (sizeof(int) * 8 - __builtin_clz(n)) {}
  // change here, calculate value given child intervals, lazy
       val, interval length
  T combine(T a, T b, L val = 0, int len = 0) { return a + b +
       val * len;  }
  void apply(int p, L val, int len) {
    if (p < n)
      lazy[p] += val; // change here, propagate val to lazy[p]
    tree[p] = combine(tree[p], def, val, len);
  }
  void init() {
    for (int i = n - 1; i > 0; i--)
      tree[i] = combine(tree[i << 1], tree[i << 1 | 1]);
  }
  void build(int p) {
    int len = 1;
    while (p > 1)
      p >>= 1, len <<= 1, tree[p] = combine(tree[p << 1], tree[
           p << 1 | 1], lazy[p], len);
  }
  void push(int p) {
    for (int s = h, len = 1 << (h - 1); s > 0; s--, len >>= 1)
         {
      int i = p >> s;
      if (lazy[i] != ldef) {
        apply(i << 1, lazy[i], len);
        apply(i << 1 | 1, lazy[i], len);
        lazy[i] = ldef;
      }
    }
  }
  // update on [l,r)
  void increment(int l, int r, L val) {
    l += n, r += n;
    push(l), push(r - 1);
    int l0 = l, r0 = r, len = 1;
    for (; l < r; l >>= 1, r >>= 1, len <<= 1) {
      if (l & 1) apply(l++, val, len);
      if (r & 1) apply(--r, val, len);
    }
    build(l0), build(r0 - 1);
  }
  // query on [l,r)
  T query(int l, int r) {
    l += n, r += n;
    push(l), push(r - 1);
    T resl = def, resr = def;
    for (; l < r; l >>= 1, r >>= 1) {
      if (l & 1) resl = combine(resl, tree[l++]);
      if (r & 1) resr = combine(tree[--r], resr);
    }
    return combine(resl, resr);
  }
};
using LST = LazySegTree<long long, long long>;

pair<int, int> interval(LST lst, int i) {
  if (i >= lst.n)
    return {i - lst.n, i - lst.n + 1};
  pair<int, int> l = interval(lst, i * 2);
  pair<int, int> r = interval(lst, i * 2 + 1);
  if (l.second != r.first)
    return {-1, -1};
  return {l.first, r.second};
}

void debug(LST lst) {
  for (int i = 1; i < 2 * lst.n; i++) {
    auto res = interval(lst, i);
    cout << i << ": [" << res.first << ", " << res.second << ")
         " << lst.tree[i];
    if (i < lst.n)
      cout << " " << lst.lazy[i];
    cout << endl;
  }
}
```

UnionFind.h

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

```
#define MAXN 100000

//can also be by rank
int parent[MAXN], sz[MAXN];

void make_set(int v) {
  parent[v] = v;
  sz[v] = 1;
}

int find_set(int v) {
  if (parent[v] == v)
    return v;
  return parent[v] = find_set(parent[v]);
}

void union_set(int a, int b) {
  a = find_set(a);
  b = find_set(b);
  if (sz[a] > sz[b]) {
    swap(a, b);
  }
  parent[a] = b;
  sz[b] += sz[a];
}
```

UnionFindDynamic.h

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

```
struct union_find {
  int n;
  vector<int> p;
  vector<int> sz;
  union_find(int n) : n(n), p(n), sz(n, 1) {
    iota(begin(p),end(p), 0);
  }
  int leader(int x) {
    if (p[x] == x)
      return x;
    return p[x] = leader(p[x]);
  }
  void unite(int x, int y) {
    int l = leader(x), s = leader(y);
    if (l == s) return;
    if (sz[s] > sz[l]) swap(s,l);
    p[s] = l, sz[l] += sz[s];
  }
};
```

Matrix.h

Description: Matrix

```
#define MOD 1000000007ll

struct Matrix {
  int n, m;
  vector<vector<long long>> vals;

  Matrix(int n, int m) : n(n), m(m), vals(n, vector<long long>(
       m)) {}

  vector<long long>& operator[](int i) {
    return vals[i];
  }

  Matrix operator*(const Matrix &other) {
    Matrix res(n, other.m);
    for (int k = 0; k < m; k++) {
      for (int i = 0; i < n; i++) {
        for (int j = 0; j < other.m; j++) {
          res[i][j] += vals[i][k] * other[k][j];
          res[i][j] %= MOD;
        }
      }
    }
    return res;
  }

  static Matrix exp(Matrix base, long long exp) {
    Matrix res(base.n, base.n);
    for (int i = 0; i < res.n; i++) {
      res[i][i] = 1;
    }
    while (exp > 0) {
      if (exp & 1) res = res * base;
      base = base * base;
      exp >>= 1;
    }
    return res;
  }
};
```

FenwickTree.h

Description: Fenwick tree for partial sum.
Time: Both operations are $\mathcal{O}(\log N)$.

```
#define SIZE 1000

long long bit[SIZE];

long long sum(int i) { // sum of [1,i]
  long long sum = 0;
  while (i > 0) {
    sum += bit[i];
    i -= (i) & -(i);
  }
  return sum;
}

void add(int i, long long delta) {
  while (i <= SIZE) {
    bit[i] += delta;
    i += (i) & -(i);
  }
}
```

FenwickTreeDynamic.h

Description: Fenwick tree with dynamic size for partial sum.
Time: Both operations are $\mathcal{O}(\log N)$.

```
struct fenwick_tree {
  int n;
  vector<ll> bits;
  fenwick_tree(int n) : n(n), bits(n+1) {}
  void update(int v, int delta) {
    for (++v; v <= n; v += v&(-v)) bits[v] += delta;
  }
  ll query(int r) {
    ll res = 0;
    for (++r; r > 0; r -= r&(-r)) res += bits[r];
    return res;
  }
  // sum of [l, r]
  ll query(int l, int r) { return query(r) - query(l-1); }
};
```

SparseTable.h

Description: Find minimum of [l,r] in static array.
Time: Build is $\mathcal{O}(N \log N)$ and query is $\mathcal{O}(1)$.

```
int ilog2(int x) {
  return 32-__builtin_clz(x)-1;
}
struct sparse_table {
  vector<vector<int>> st;
  sparse_table(const vector<int>& a) : st(1, a) {
    for (int i=0;i<ilog2(a.size());i++){
      st.emplace_back(a.size());
      for (int j=1<<i;j<a.size();j++) {
        st[i+1][j] = min(st[i][j], st[i][j-(1<<i)]);
      }
    }
  }
  // query on [l, r]
  int query(int l, int r) {
    int i = ilog2(r-l+1);
    return min(st[i][l+(1<<i)-1], st[i][r]);
  }
};
```

MoQueries.h

Description: Mo's algorithms that can process offline queries if boundary addition/removal to range is defined.
Time: Build is $\mathcal{O}\left((N+Q)\sqrt{N}\right)$.

```
struct mo_query {
  using func = function<void(int)>;
  int n;
  vector<pair<int, int>> lr;
  mo_query(int n) : n(n) {}
  // add query on [l, r]
  void add(int l, int r) {
    lr.emplace_back(l, r+1);
  }

  void build(const func &add_left, const func &add_right, const
      func &erase_left, const func &erase_right, const func &
     out) {
    int q = (int) lr.size();
    int bs = n / min<int>(n, (int)sqrt(q));
    vector<int> ord(q);
    iota(begin(ord), end(ord), 0);
    sort(begin(ord), end(ord), [&](int a, int b) {
      int ablock = lr[a].first / bs, bblock = lr[b].first / bs;
      if(ablock != bblock) return ablock < bblock;
      return (ablock & 1) ? lr[a].second > lr[b].second : lr[a
          ].second < lr[b].second;
    });
    int l = 0, r = 0;
    for(auto idx : ord) {
      while(l > lr[idx].first) add_left(--l);
      while(r < lr[idx].second) add_right(r++);
      while(l < lr[idx].first) erase_left(l++);
      while(r > lr[idx].second) erase_right(--r);
      out(idx);
    }
  }

  void build(const func &add, const func &erase, const func &
       out) {
    build(add, add, erase, erase, out);
  }
};
```

Numerical (4)

4.1 Fourier transforms

NumberTheoreticTransform.h

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2.
Time: $\mathcal{O}(N \log N)$

```
#define MOD 998244353
template<long long P>
struct NTTHelper {
  static const int sz = 1 << 20;
  long long omega[sz];

  NTTHelper() {
    omega[sz / 2] = 1;
    long long exp = 1, base = 3, pow = P / sz;
    while (pow) {
      if (pow & 1)
        exp = exp * base % P;
      base = base * base % P;
      pow >>= 1;
    }
    for (int i = sz / 2 + 1; i < sz; i++)
      omega[i] = omega[i - 1] * exp % P;
    for (int i = sz / 2 - 1; i > 0; i--)
      omega[i] = omega[i << 1];
  }

  void ntt(long long *arr, int m) {
    if (m == 1)
      return;
    ntt(arr, m / 2);
    ntt(arr + m / 2, m / 2);
    for (int i = 0; i < m / 2; i++) {
      long long e = arr[i], o = omega[i + m / 2] * arr[i + m /
          2] % P;
      arr[i] = e + o < P ? e + o : e + o - P;
      arr[i + m / 2] = e - o >= 0 ? e - o : e - o + P;
    }
  }

  void ntt(vector<long long> &arr, bool inverse) {
    int m = arr.size();
    for (int i = 1, j = 0; i < m; i++) {
      int bit = m >> 1;
      for (; j & bit; bit >>= 1)
        j ^= bit;
      j ^= bit;

      if (i < j)
        swap(arr[i], arr[j]);
    }
    ntt(arr.data(), m);
    if (inverse)
    {
      reverse(arr.begin() + 1, arr.end());
      for (int i = 0; i < m; i++)
        arr[i] = arr[i] * (P - P / m) % P;
    }
  }

  vector<long long> multiply(vector<long long> a, vector<long
      long> b) {
    ntt(a, false);
    ntt(b, false);
    vector<long long> res(a.size());
    for (int i = 0; i < res.size(); i++) {
      res[i] = a[i] * b[i] % P;
    }
    ntt(res, true);
    return res;
  }
};
NTTHelper<MOD> helper;

vector<long long> multiply(vector<long long> a, vector<long
    long> b) {
  int sz = 1 << (sizeof(int) * 8 - __builtin_clz(a.size() + b.
      size() - 2));
  a.resize(sz), b.resize(sz);
  vector<long long> res = helper.multiply(a, b);
  res.resize(a.size() + b.size() - 1);
  return res;
}
```

NumberTheoreticTransformArbitrary.h

Description: Arbitrary mod ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. MOD is $4 * 10^{17}$ for default setting.
Time: $\mathcal{O}(N \log N)$

```
const long long MOD1 = 998244353ll; // 2^23 * 7 * 17 + 1
const long long MOD2 = 469762049ll; // 2^26 * 7 + 1
// 167772161 = 2^25 * 5 + 1
// 7340033 = 2^20 * 7 + 1
// 3 is primitive root for all
NTTHelper<MOD1> helper1;
NTTHelper<MOD2> helper2;

vector<long long> multiply(vector<long long> a, vector<long
    long> b) {
  int sz = 1 << (sizeof(int) * 8 - __builtin_clz(a.size() + b.
      size() - 2));
  a.resize(sz), b.resize(sz);
  vector<long long> res1 = helper1.multiply(a, b);
  vector<long long> res2 = helper2.multiply(a, b);
  res1.resize(a.size() + b.size() - 1), res2.resize(a.size() +
      b.size() - 1);

  __int128_t p1 = 260520730147305702, p2 = 208416582520653596,
      mod = (__int128_t)MOD1*MOD2;
  vector<long long> res(res1.size());
  for (int i = 0; i < res.size(); i++) {
    res[i] = (res1[i] * p1 + res2[i] * p2) % mod;
  }
  return res;
```

```
}
```

FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x]\exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution

Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (~1s for $N = 2^{22}$)

```
using cd = complex<double>;

const double PI = atan(1) * 4;

struct FFTHelper {
  static const int sz = 1 << 20;
  cd omega[sz];
  FFTHelper() {
    omega[sz / 2] = 1;
    cd pow = exp(cd(0, 2 * PI / sz));
    for (int i = sz / 2 + 1; i < sz; i++)
      omega[i] = omega[i - 1] * pow;
    for (int i = sz / 2 - 1; i > 0; i--)
      omega[i] = omega[i << 1];
  }

  void fft(cd *arr, int m) {
    if (m == 1)
      return;
    fft(arr, m / 2);
    fft(arr + m / 2, m / 2);
    for (int i = 0; i < m / 2; i++) {
      cd e = arr[i], o = omega[i + m / 2] * arr[i + m / 2];
      arr[i] = e + o;
      arr[i + m / 2] = e - o;
    }
  }

  void fft(vector<cd> &arr, bool inverse) {
    int m = arr.size();
    for (int i = 1, j = 0; i < m; i++) {
      int bit = m >> 1;
      for (; j & bit; bit >>= 1)
        j ^= bit;
      j ^= bit;

      if (i < j)
        swap(arr[i], arr[j]);
    }
    fft(arr.data(), m);
    if (inverse) {
      reverse(arr.begin() + 1, arr.end());
      cd inv((double)1 / m, 0);
      for (int i = 0; i < m; i++)
        arr[i] = arr[i] * inv;
    }
  }

  vector<cd> multiply(vector<cd> a, vector<cd> b) {
    fft(a, false);
    fft(b, false);
    vector<cd> res(a.size());
    for (int i = 0; i < res.size(); i++) {
      res[i] = a[i] * b[i];
    }
    fft(res, true);
    return res;
  }
} helper;
```

4.2 Matrix

RREF.h

Description: Row reduce matrix.

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

```
const ld eps = 1e-9;
int rref(vector<vector<ld>> &a) {
  int n = a.size();
  int m = a[0].size();
  int r = 0;
  for (int c = 0; c < m && r < n; c++) {
    int j = r;
    for (int i = r + 1; i < n; i++)
      if (fabs(a[i][c]) > fabs(a[j][c])) j = i;
    if (fabs(a[j][c]) < eps) continue;
    swap(a[j], a[r]);

    ld s = 1.0 / a[r][c];
    for (int j = 0; j < m; j++) a[r][j] *= s;
    for (int i = 0; i < n; i++) if (i != r) {
      ld t = a[i][c];
      for (int j = 0; j < m; j++) a[i][j] -= t * a[r][j];
    }
    r++;
  }
  return r;
}
```

XorBasis.h

Description: Find xor basis of numbers. Useful for finding number of distinct integers represetned by xor of subset or maximum xor of subset.

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

```
struct xor_basis {
  int sz = 0;
  vector<int> basis; // basis vector with ith bit set
  xor_basis(int n) : basis(n) {}
  void insert(int mask) {
    for (int i = 0; i < basis.size(); i++) {
      if ((mask & 1 << i) == 0) continue;
      if (basis[i]) {
        basis[i] = mask;
        ++sz;
        return;
      }
      mask ^= basis[i];
    }
  }
};
```

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

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

```
const int MOD = 1000000007; // 998244353
struct mint {
  int x;
  mint(int x = 0) : x(norm(x)) { }
  mint(ll x) : x(norm(x%MOD)) { }
  int norm(int x) const { if (x < 0) { x += MOD; } if (x >= MOD
      ) { x -= MOD; } return x; }
  mint operator+(mint o) const { return norm(x + o.x); }
  mint operator-(mint o) const { return norm(x - o.x); }
  mint operator*(mint o) const { return ll(x) * o.x; }
  mint operator/(mint o) { return *this * o.inv(); }
  mint inv() const { return exp(MOD - 2); }
  mint exp(ll n) {
    if (!n) return mint(1);
    mint a = *this;
    mint r = a.exp(n / 2); r = r * r;
    return n&1 ? a * r : r;
  }
};
```

ModPow.h

```
ll binexp(ll b, ll n) {
  ll res = 1;
  for (; n; b = b * b % MOD, n /= 2)
    if (n & 1) res = res * b % MOD;
  return res;
}
```

5.2 Primality

Eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit.

```
#define SIEVE_SIZE 10000000

bool isPrime[SIEVE_SIZE+1];
int spf[SIEVE_SIZE+1]; //smallest prime factor
vector<int> primes;

void sieve() {
  fill(begin(isPrime) + 2, end(isPrime), true);
  for (int i = 2; i <= SIEVE_SIZE; i++) {
    if (isPrime[i]) {
      primes.push_back(i);
      spf[i] = i;
    }
    for (int j = 0; j < primes.size() && i * primes[j] <=
        SIEVE_SIZE && primes[j] <= spf[i]; j++) {
      isPrime[i * primes[j]] = false;
      spf[i * primes[j]] = primes[j];
    }
  }
}
```

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"

```
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"

```
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;
}
```

5.3 Divisibility

euclid.h

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

```
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 $0 \le x < \mathrm{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$

"euclid.h"

```
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;
}
```

5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

Description: *Euler's* ϕ function is defined as $\phi(n) := \#$ of positive integers $\le n$ that are coprime with n. This template can be used to calculate any multiplicative function (e.g. mobius) $\phi(1) = 1$, 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 $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.

```
vector<int>prime;
bool is_prime[MAXN];
int phi[MAXN];

void sieve () {
  fill(is_prime + 2, is_prime + MAXN, true);
  phi[1] = 1;
  for (int i = 2; i < MAXN; ++i) {
    if (is_prime[i]) {
      prime.push_back (i);
      phi[i] = i - 1; //i is prime
    }
    for (int j = 0; j < prime.size () && i * prime[j] < MAXN;
        ++j) {
      is_prime[i * prime[j]] = false;
      if (i % prime[j] == 0) {
        // use transition
        phi[i * prime[j]] = phi[i] * prime[j];  // prime[j]
            divides i
        break;
      } else {
        // use multiplicative property
        phi[i * prime[j]] = phi[i] * phi[prime[j]]; //prime[j]
            does not divide i
      }
    }
  }
}
```

5.4 Estimates

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

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

5.5 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)

e.g. $[gcd(a, b) = 1] = \sum_{d|gcd(a,b)} \mu(d)$

Count coprime pairs $\sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor^2$

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Cycles

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

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

6.1.2 Derangements

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

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

6.1.3 Burnside's lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Lucas' Theorem

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

6.3 General purpose numbers

6.3.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.2 Eulerian numbers

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

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

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

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

6.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.4 Labeled unrooted trees

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

\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$

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

6.3.5 Catalan numbers

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

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

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

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

Graph (7)

7.1 Fundamentals

BellmanFord.h

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

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

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

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

7.2 Network flow

MinCostMaxFlow.h

Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges are not. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.
Time: Approximately $\mathcal{O}(E^2)$

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

const ll INF = numeric_limits<ll>::max() / 4;
typedef vector<ll> VL;

struct MCMF {
  int N;
  vector<vi> ed, red;
  vector<VL> cap, flow, cost;
  vi seen;
  VL dist, pi;
  vector<pii> par;

  MCMF(int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
  }

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

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

    auto relax = [&](int i, ll cap, ll cost, int dir) {
      ll val = di - pi[i] + cost;
      if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i], i});
        else q.modify(its[i], {-dist[i], i});
      }
    };

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (int i : ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
      for (int i : red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0);
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x][p]);
      totflow += fl;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        if (r) flow[p][x] += fl;
        else flow[x][p] -= fl;
    }
    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
  }

  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (int to : ed[i]) if (cap[i][to])
          if ((v = pi[i] + cost[i][to]) < pi[to])
            pi[to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

7.3 Matching

BipariateMatch.h

Description: Bipariate matching with dfs trying to invert all augmenting path.
Time: $\mathcal{O}(NM)$

```
struct bipariate_match {
  int n,m;
  vector<int> mt;
  vector<vector<int>> adj;
  bipariate_match(int n, int m) : n(n), m(m), mt(m, -1), adj(n,
        vector<int>()) {}
  void add(int u, int v) {
    adj[u].push_back(v);
  }
  void run() {
    vector<bool> vis;
    auto dfs = [&](auto self, int u) -> bool {
      if (vis[u]) return false;
      vis[u] = true;
      for (int v : adj[u]) {
        if (mt[v] == -1 || self(self, mt[v])) {
          mt[v] = u;
          return true;
        }
      }
```

```
      return false;
    };

    // arbitrary matching heuristics
    vector<bool> used(n);
    for (int i=0;i<n;i++){
      for (int v : adj[i]) {
        if (mt[v] == -1) {
          mt[v] = i;
          used[i] = true;
          break;
        }
      }
    }

    for (int i=0;i<n;i++) {
      // if implicit graph needs match[i] == -1
      if (used[i]) continue;
      vis.assign(n, false);
      dfs(dfs, i);
    }
  }
};
```

7.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph.

```
struct sc_components {
  int n;
  vector<vector<int>> adj, adj_inv;
  vector<vector<int>> comps;
  vector<int> comp_ids;
  int num_comp = 0;
  sc_components(int n) : n(n), adj(n), adj_inv(n) {}

  // add edge
  void add(int u, int v) {
    adj[u].push_back(v);
    adj_inv[v].push_back(u);
  }

  // find strongly connected components of graph
  void run() {
    vector<bool> vis(n);
    vector<int> order;
    auto topo_dfs = [&](auto self, int u) -> void {
      vis[u] = true;
      for (int v : adj[u]) if (!vis[v]) self(self, v);
      order.push_back(u);
    };
    for (int i=0;i<n;i++) if(!vis[i]) topo_dfs(topo_dfs, i);
    reverse(begin(order), end(order));
    comp_ids.assign(n, 0);
    vis.assign(n, false);
    auto comp_dfs = [&](auto self, int u) -> void {
      vis[u] = true;
      comp_ids[u] = num_comp;
      comps.back().push_back(u);
      for (int v : adj_inv[u]) if (!vis[v]) self(self, v);
    };
    for (int i : order)
      if (!vis[i]) {
        comps.push_back({});
        comp_dfs(comp_dfs, i);
        num_comp++;
      }
  }
};
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable.

Usage: `TwoSat ts(number of boolean variables);`
`ts.either(0, ~3); // Var 0 is true or var 3 is false`

Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

"SCC.h"

```
struct two_sat {
  int n;
  vector<int> res;
  sc_components scc;

  two_sat(int n) : n(n), scc(2*n) {}

  int inv(int v) {
    if (v < n)
      return n + v;
    else
      return v - n;
  }

  int norm(int v){
    if (v >= n)
      return v - n;
    return v;
  }

  // a or b <=> (~b => a and ~a => b)
  void either(int a, int b) {
    scc.add(inv(a), b);
    scc.add(inv(b), a);
  }

  bool run() {
    scc.run();

    res.assign(n, -1);
    for (int i=scc.num_comp-1;i>=0;i--){
      for (int a : scc.comps[i]) {
        if (scc.comp_ids[a] == scc.comp_ids[inv(a)]) {
          return false;
        }
        if (res[norm(a)] == -1)
          res[norm(a)] = a < n;
      }
    }
    return true;
  }
};
```

7.5 Trees

BinaryLifting.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.

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

```
vector<int> adj[MAXN];
int parent[MAXN][20], depth[MAXN];

int getParent(int x, int k) {
  for (int j = 0; k >= 1 << j; j++) {
    if (k & (1 << j))
      x = parent[x][j];
  }
  return x;
}

void dfs(int cur) {
  for (int next : adj[cur]) {
    if (next == parent[cur][0])
      continue;
    parent[next][0] = cur;
    depth[next] = depth[cur] + 1;
    dfs(next);
  }
}

int main() {
  int n, q;
  cin >> n >> q;

  for (int i = 0; i < n - 1; i++) {
    int a, b;
    cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  dfs(1);

  for (int i = 1; i <= 18; i++) {
    for (int j = 1; j <= n; j++) {
      parent[j][i] = parent[parent[j][i - 1]][i - 1];
    }
  }

  for (int i = 0; i < q; i++) {
    int a, b;
    cin >> a >> b;

    if (depth[a] > depth[b])
      swap(a, b);

    int res = depth[b] - depth[a];
    b = getParent(b, res);

    if (a == b) {
      cout << res << endl;
      continue;
    }

    for (int j = 18; j >= 0; j--) {
      if (parent[a][j] != parent[b][j]) {
        res += (2 << j);
        a = parent[a][j];
        b = parent[b][j];
      }
    }

    // res is distance between a and b
    cout << res + 2 << endl;
    // parent[a][0] is LCA
    cout << parent[a][0] << endl;
  }
}
```

BinaryLiftingDynamic.h

Description: Binary lifting with dynamic size;

```
vector<vector<int>> treeJump(vector<int>& P){
  int d = ceil(log2(P.size()));
  vector<vector<int>> up(d, P);
  for(int i=1;i<d;i++) for(int j=0;j<P.size();j++)
    up[i][j] = up[i-1][up[i-1][j]];
  return up;
```

```
}

int jmp(vector<vector<int>>& up, int node, int steps){
  for(int i=0;i<up.size();i++)
    if(steps&(1<<i)) node = up[i][node];
  return node;
}

int lca(vector<vector<int>>& up, vector<int>& depth, int a, int
     b) {
  if (depth[a] < depth[b]) swap(a, b);
  a = jmp(up, a, depth[a] - depth[b]);
  if (a == b) return a;
  for (int i = up.size()-1;i>=0;i--) {
    int c = up[i][a], d = up[i][b];
    if (c != d) a = c, b = d;
  }
  return up[0][a];
}
```

Centroid.h

Description: Find centroid of tree.

```
int n, sz[MAXN];
vector<int> adj[MAXN];

void dfsSize(int cur, int prev) {
  sz[cur] = 1;
  for (int next : adj[cur]) {
    if (next == prev)
      continue;
    dfsSize(next, cur);
    sz[cur] += sz[next];
  }
}

int getCentroid(int cur, int prev) {
  for (int next : adj[cur]) {
    if (next == prev)
      continue;
    if (sz[next] * 2 > n)
      return getCentroid(next, cur);
  }
  return cur;
}
```

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q)$

"../data-structures/SparseTable.h"

```
struct LCA {
  int T = 0;
  vector<int> time, path, ret;
  sparse_table rmq;

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

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.
Time: $\mathcal{O}\left((\log N)^2\right)$

"../data-structures/SegmentTree.h"

```
vector<int> adj[MAXN];
int vals[MAXN];
// pos is position in heavy[i] is heavy child of i
int parent[MAXN], heavy[MAXN], pos[MAXN], depth[MAXN], head[
    MAXN];
int curPos = 0;

int dfs(int cur) {
  int size = 1, heavySize = 0;
  for (int next : adj[cur]) {
    if (next == parent[cur])
      continue;
    parent[next] = cur;
    int sz = dfs(next);
    if (sz > heavySize) {
      heavySize = sz;
      heavy[cur] = next;
    }
    size += sz;
  }
  return size;
}

void decompose(int cur, int h) {
  head[cur] = h;
  pos[cur] = curPos++;
  if (heavy[cur] != 0) {
    depth[heavy[cur]] = depth[cur];
    decompose(heavy[cur], h);
  }
  for (int next : adj[cur]) {
    if (next == parent[cur] || next == heavy[cur])
      continue;
    depth[next] = depth[cur] + 1;
    decompose(next, next);
  }
}

int main() {
  int n, q;
  cin >> n >> q;

  for (int i = 1; i <= n; i++)
    cin >> vals[i];

  for (int i = 1; i < n; i++) {
    int a, b;
    cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  dfs(1);
  decompose(1, 1);

  ST st(n);
  for (int i = 1; i <= n; i++) {
    st.tree[n + pos[i]] = vals[i];
  }
  st.init();

  for (int i = 0; i < q; i++) {
    int inp, a, b;
    cin >> inp >> a >> b;
    if (inp == 1) {
      st.update(pos[a], b);
    } else {
      int ans = 0;
      while (head[a] != head[b]) {
        if (depth[a] < depth[b])
          swap(a, b);
        ans = max(ans, st.query(pos[head[a]], pos[a] + 1));
        a = parent[head[a]];
      }
      if (pos[a] > pos[b])
        swap(a, b);
      ans = max(ans, st.query(pos[a], pos[b] + 1));
      cout << ans << endl;
    }
  }
}
```

Strings (8)

8.1 Strings

PrefixFunction.h

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

```
vector<int> prefix_function(string s) {
  int n = s.size();
  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;
}
```

ZFunction.h

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

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

StringMath.h

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

```
string removeLeadingZeros(string a) {
  for (int i = 0; i < a.length(); i++) {
    if (a[i] != '0' || i == a.length() - 1) {
      return a.substr(i);
    }
  }
  return a;
}

string add(string a, string b) {
  string res = "";
  int sum = 0;
  for (int i = 0; i < max(a.length(), b.length()); i++) {
    if (i < a.length())
      sum += a[a.length() - 1 - i] - '0';
    if (i < b.length())
      sum += b[b.length() - 1 - i] - '0';

    res = (char)((sum % 10) + '0') + res;

    sum /= 10;
  }
  if (sum == 1)
    res = "1" + res;

  return removeLeadingZeros(res);
}

string multiply(string a, string b) {
  string res = "0";
  for (int j = 0; j < b.length(); j++) {
    // multiply a by digit of b
    string curDig = "";
    for (int i = 0; i < j; i++) {
      curDig += "0";
    }

    int carry = 0;
    for (int i = 0; i < a.length(); i++) {
      carry += (a[a.length() - 1 - i] - '0') * (b[b.length() -
           1 - j] - '0');

      curDig = (char)((carry % 10) + '0') + curDig;

      carry /= 10;
    }
    if (carry != 0)
      curDig = (char)((carry % 10) + '0') + curDig;

    res = add(res, curDig);
  }

  return removeLeadingZeros(res);
}
```

SuffixArray.h

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

```
int order[2][MAXN], classes[2][MAXN], cnt[MAXN];
int n;

// sorts order by value in class
void countSort(int classCount) {
  fill(cnt, cnt + classCount, 0);
  for (int i = 0; i < n; i++) {
    cnt[classes[0][i]]++;
  }
  for (int i = 1; i < classCount; i++) {
    cnt[i] += cnt[i - 1];
  }
  for (int i = n - 1; i >= 0; i--) {
    order[1][--cnt[classes[0][order[0][i]]]] = order[0][i];
  }
  swap(order[0], order[1]);
}

int main() {
  string str;
  cin >> str;
  n = str.length();

  int classCount = 26;
  for (int i = 0; i < n; i++) {
    order[0][i] = i;
    classes[0][i] = str[i] - 'a';
  }

  for (int i = -1, shift = 0; shift == 0 || (1 << i) < n;
       i++, shift = (1 << i)) {
    for (int j = 0; j < n; j++) {
      order[0][j] -= shift;
      if (order[0][j] < 0)
        order[0][j] += n;
    }
    countSort(classCount);
    classes[1][order[0][0]] = 0;
    classCount = 1;
    for (int j = 0; j < n; j++) {
      pair<int, int> cur = {classes[0][order[0][j]],
                            classes[0][(order[0][j] + shift) %
                                   n]};
      pair<int, int> prev = {classes[0][order[0][j - 1]],
                             classes[0][(order[0][j - 1] +
                                    shift) % n]};
      if (cur != prev)
        classCount++;
      classes[1][order[0][j]] = classCount - 1;
    }
    swap(classes[0], classes[1]);
  }

  cout << str.substr(order[0][0]) << str.substr(0, order[0][0])
       << endl;
}
```

SuffixTree.h

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

```
string str;
Node* root;
Node* active_node;
int active_length;
int rem; //remainder

struct Node {
  int l, r;
  Node *parent, *link;
  unordered_map<char, Node *> next;

  Node(int l = 0, int r = 0, Node *parent = nullptr)
      : l(l), r(r), parent(parent), link(nullptr) {
    if (parent != nullptr) {
      parent->next[str[l]] = this;
    }
  }
  Node *getNext(char c) {
    if (!next.count(c))
      return nullptr;
    return next[c];
  }
  char getChar(int length) {
    if (r == 0)
      return '\0';
    return str[l + length];
  }
};

void add(int pos) {
  rem++;

  if (active_node->getChar(active_length) != str[pos]) {
    Node *prev = nullptr;
    while (rem > 0) {
      if (active_length != 0) {
        Node *split = new Node(active_node->l, pos, active_node
            ->parent);
        // move active node to non-overlapping portion
        active_node->l += active_length;
        // update edge so split is parent of active_node
        active_node->parent = split;
        split->next[str[active_node->l]];

        active_node = split;
        active_length--;

        if (prev != nullptr) {
          prev->link = split;
        }
        prev = split;
      }
      active_node->next[str[pos]] = new Node(pos, str.length(),
           active_node);
      rem--;

      if (active_node->parent == root) {
        active_node = root->next[active_node->getChar(1)];
      } else {
        active_node = active_node->link->next[active_node->
            getChar(1)];
      }
    }
  } else {
    active_length++;
  }
}

int main() {
  root = new Node();

  active_length = 0;
  active_node = root;
  rem = 0;
}
```

Trie.h

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

```
struct Node {
  Node *children[26];

  bool isEnd;
};

// creates/initializes a new node
Node *getNode() {
```

```
  Node *pNode = new Node;
  for (int i = 0; i < 26; i++) {
    pNode->children[i] = nullptr;
  }
  pNode->isEnd = false;
  return pNode;
}

void ins(Node *r, string word) {
  Node *curNode = r;
  for (int i = 0; i < word.length(); i++) {
    if (curNode->children[word[i] - 'a'] == nullptr)
      curNode->children[word[i] - 'a'] = getNode();
    curNode = curNode->children[word[i] - 'a'];
  }

  curNode->isEnd = true;
}

// search function not written because it was not needed for
    the problem

// single struct
struct Node {
  Node *children[26];
  bool isEnd;

  // create empty node, also used to create root
  static Node *getNode() {
    Node *pNode = new Node;
    for (int i = 0; i < 26; i++) {
      pNode->children[i] = nullptr;
    }
    pNode->isEnd = false;
    return pNode;
  }

  // should only be used by root
  void insert(string word) {
    Node *curNode = this;
    for (int i = 0; i < word.length(); i++) {
      if (curNode->children[word[i] - 'a'] == nullptr)
        curNode->children[word[i] - 'a'] = getNode();
      curNode = curNode->children[word[i] - 'a'];
    }
    curNode->isEnd = true;
  }
};
```

LongestPalindrome.h

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

```
int lpr[MAXN * 2];

int main() {
  string str;
  cin >> str;

  string s = "|";

  for (int i = 0; i < str.length(); i++) {
    s += str[i];
    s += '|';
  }

  int center = 0;
  int radius = 0;

  while (center < s.length()) {
    while (center - radius >= 0 && center + radius < s.length()
        &&
         s[center - radius] == s[center + radius]) {
      radius++;
    }
    radius--;
    lpr[center] = radius;

    int oldCenter = center;
    int oldRadius = radius;
    center++;

    while (center <= oldCenter + oldRadius) {
      int mirroredCenter = oldCenter - (center - oldCenter);
      int largestRadius = oldCenter + oldRadius - center;
      if (lpr[mirroredCenter] < largestRadius) {
        lpr[center] = lpr[mirroredCenter];
        center++;
      } else if (lpr[mirroredCenter] > largestRadius) {
        lpr[center] = largestRadius;
        center++;
      } else {
        radius = largestRadius;
        break;
      }
    }
  }

  int maxi = 0;
  for (int i = 0; i < s.length(); i++) {
    if (lpr[i] > lpr[maxi])
      maxi = i;
  }

  for (int i = maxi - lpr[maxi] + 1; i <= maxi + lpr[maxi] - 1;
       i += 2) {
    cout << s[i];
  }
  cout << endl;
}
```

MinimalRotation.h

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

```
int prefix[MAXN * 2];

int booth(string s) {
  s += s;
  // best rotation so far
  int rot = 0;
  for (int r = 1; r < s.length(); r++) {
    int l = prefix[r - 1 - rot];
    while (l > 0 && s[l + rot] != s[r]) {
      if (s[l + rot] > s[r]) {
        rot = r - l;
      }
      l = prefix[l - 1];
    }
    if (l == 0 && s[l + rot] != s[r]) {
      if (s[l + rot] > s[r]) {
         rot = r;
      }
      prefix[r - rot] = 0;
    }
    else {
      prefix[r - rot] = l + 1;
    }
  }
  return rot;
}
```

Geometry (9)

9.1 Geometric primitives

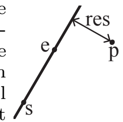
Point.h

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

```
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()); }
  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)); }
};
```

lineDistance.h

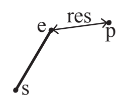
Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate 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"

```
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.

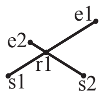
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h"

```
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.



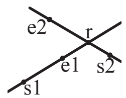
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

"Point.h", "OnSegment.h"

```
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 {begin(s), end(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"

```
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"

```
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"

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

9.2 Circles

CircleIntersection.h

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

"Point.h"

```
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;
}
```

CirclePolygonIntersection.h

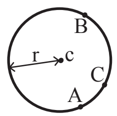
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h"

```
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;
  for(int i=0;i<ps.size();i++)
    sum += tri(ps[i] - c, ps[(i + 1) % ps.size()] - 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"

```
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;
}
```

9.3 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"

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = p.size();
  for(int i=0;i<n;i++) {
    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"

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

PolygonCenter.h

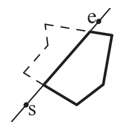
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h"

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = v.size() - 1; i < v.size(); 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.

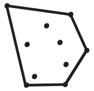


Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h"

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  for(int i=0;i<poly.size();i++) {
    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;
}
```

ConvexHull.h



Description: Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$

"Point.h"

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (pts.size() <= 1) return pts;
  sort(pts.begin(), pts.end());
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(pts.begin(), pts.end
       ()))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h"

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = S.size(), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  for (int i=0;i<j;i++)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h"

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (l.size() < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

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

"Point.h"

```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = poly.size(), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

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

9.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h"

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(v.size() > 1);
  set<P> S;
  sort(v.begin(), v.end(), [](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;
}
```

Debugging utils

debug.cpp

14 lines

```
#define dbg(x) "(" << #x <<": " << (x) << ") "

template<typename ...Ts>
ostream& operator<<(ostream& os, const pair<Ts...>& p){
  return os<<"{"<<p.first<<", "<<p.second<<"}";
}

template<typename Ostream, typename Cont>
enable_if_t<is_same_v<Ostream,ostream>, Ostream&>
operator<<(Ostream& os, const Cont& v){
  os<<"[";
  for(auto& x:v){os<<x<<", ";}
  return os<<"]";
}
```

Print int128

print.cpp

8 lines

```
void print(__int128 x) {
  if (x < 0) {
    cout<<'-';
    x = -x;
  }
  if (x > 9) print(x / 10);
  cout<<(int)(x % 10);
}
```