Birla Institute of Technology and Science, Pilani Campus

lite lo

Tanveer Singh, Anish Kasegaonkar, Utkarsh Darolia

ICPC Asia-West Final Contest

2024-03-28

Contest (1)

template.cpp

17 lines

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'

#define rep(i, a, b) for(int i = a; i < (
   b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int32_t main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
  cout << setprecision(20) << fixed;
}
```

.bashrc

3 lines

```
alias c='g++ -Wall -g -std=c++17 -
   fsanitize=undefined,address'
alias i='./a.out < input.txt'
alias io='./a.out < input.txt > output.
   txt'
```

Data structures (2)

OrderStatisticTree.h

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

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> b43fbb, 4 lines

```
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,
   less<int>, rb_tree_tag,
   tree_order_statistics_node_update>
/* 1. find_by_order(k) : Returns an
   iterator to the kth element (counting
    from zero) in the set in O(Log N)
   time.
2. order_of_key(k) : Returns the number
   of items that are strictly smaller
   than the item K in O(Log N) time. */
```

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).

ba52e5, 6 lines

```
const int RANDOM = chrono::
   high_resolution_clock::now().
   time_since_epoch().count();
struct chash {
  const uint64_t C = ll(4e18 * acos(0)) |
     71; // large odd number
  ll operator()(ll x) const { return
     __builtin_bswap64((x^RANDOM)*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h
   ({},{},{},{},{1<<16});
```

LazySegmentTree.h

Description: T stores segment tree data, U stores lazy data. Change SID, LID, comb and push based on implementation.
Time: $\mathcal{O}(\log N)$.

731a64, 55 lines

```
template <class T, class U>
struct LazySegTree {
  int n;
  vector<T> t; vector <U> lz;
  T SID = 0;   // Identity element for
     segtree data
  U LID = 0;  // Identity element for
     lazy update

  T comb(T a, T b) {  // Segtree function
    return a + b;
  }
void push(int node, int l, int r) {  //
    Propagation
  t[node] += (r - l + 1)*lz[node];
  if(l != r) rep(it, 0, 2) {
    z[node*2 + it] += lz[node];
  }
  lz[node] = LID;
}
LazySegTree(int _n) : n(_n) {
  t.resize(4*n, SID);
  lz.resize(4*n, LID);
}
void build(int node, int l, int r,
   vector<T> &v) {
  if(l == r) {
    t[node] = v[l];
  } else {
    int m = (l + r)/2;
    build(node*2, l, m, v);
    build(node*2+1, m+1,r,v);
    pull(node);
  }
}
void pull(int node) { t[node] = comb(t
   [2 * node], t[2 * node + 1]); }
void apply(int l, int r, U val) { apply
   (l, r, val, 1, 0, n - 1); }
void apply(int l, int r, U val, int
   node, int tl, int tr) {
  push(node, tl, tr);
  if (r < tl || tr < l) return;
  if (l <= tl && tr <= r) {
  lz[node] = val; // Set value here.
    push(node, tl, tr);
    return;
  }
  int tm = (tl + tr) / 2;
  apply(l, r, val, 2 * node, tl, tm);
  apply(l, r, val, 2 * node + 1, tm +
     1, tr);
  pull(node);
}
```

```
  T query(int l, int r) { return query(l,
      r, 1, 0, n - 1); }
  T query(int l, int r, int node, int tl,
      int tr) {
    push(node, tl, tr);
    if (r < tl || tr < l) return SID;
    if (l <= tl && tr <= r) return t[node
        ];
    int tm = (tl + tr) / 2;
    return comb(query(l, r, 2 * node, tl,
         tm), query(l, r, 2 * node + 1,
        tm + 1, tr));
  }
};
```

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;
  }
```

```
};
```

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;
  }
};
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$

9556fc, 55 lines

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

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

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

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

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

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

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

2DSegTree.h

Description: 2D SegTree

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

617302, 66 lines

```
// a[n][m] is the given array
// t[4*n+1][4*m+1] is the segment tree
   array to be made
void build_y(int vx, int lx, int rx, int
   vy, int ly, int ry) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = a[lx][ly];
        else
            t[vx][vy] = t[vx*2][vy] + t[
               vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        build_y(vx, lx, rx, vy*2, ly, my)
           ;
        build_y(vx, lx, rx, vy*2+1, my+1,
            ry);
        t[vx][vy] = t[vx][vy*2] + t[vx][
           vy*2+1];
    }
}
void build_x(int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x(vx*2, lx, mx);
        build_x(vx*2+1, mx+1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m-1);
}
int sum_y(int vx, int vy, int tly, int
   try_, int ly, int ry) {
    if (ly > ry)
        return 0;
    if (ly == tly && try_ == ry)
        return t[vx][vy];
    int tmy = (tly + try_) / 2;
    return sum_y(vx, vy*2, tly, tmy, ly,
       min(ry, tmy))
         + sum_y(vx, vy*2+1, tmy+1, try_,
            max(ly, tmy+1), ry);
}
int sum_x(int vx, int tlx, int trx, int
   lx, int rx, int ly, int ry) {
    if (lx > rx)
        return 0;
    if (lx == tlx && trx == rx)
        return sum_y(vx, 1, 0, m-1, ly,
           ry);
    int tmx = (tlx + trx) / 2;
    return sum_x(vx*2, tlx, tmx, lx, min(
       rx, tmx), ly, ry)
         + sum_x(vx*2+1, tmx+1, trx, max(
            lx, tmx+1), rx, ly, ry);
}
void update_y(int vx, int lx, int rx, int
    vy, int ly, int ry, int x, int y,
   int new_val) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = new_val;
        else
            t[vx][vy] = t[vx*2][vy] + t[
               vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        if (y <= my)
            update_y(vx, lx, rx, vy*2, ly
               , my, x, y, new_val);
        else
            update_y(vx, lx, rx, vy*2+1,
               my+1, ry, x, y, new_val);
        t[vx][vy] = t[vx][vy*2] + t[vx][
           vy*2+1];
    }
}
void update_x(int vx, int lx, int rx, int
    x, int y, int new_val) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        if (x <= mx)
            update_x(vx*2, lx, mx, x, y,
               new_val);
        else
            update_x(vx*2+1, mx+1, rx, x,
                y, new_val);
    }
    update_y(vx, lx, rx, 1, 0, m-1, x, y,
        new_val);
}
```

RMQ.h

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

510c32, 16 lines

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

MoQueries.h

Description: Answer interval queries by smart ordering of queries.
Time: $\mathcal{O}\left(N\sqrt{Q}\right)$

7b2870, 20 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;
}
```

Numerical (3)

3.1 Polynomials and recurrences

PolyInterpolate.h

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

08bf48, 13 lines

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

BerlekampMassey.h

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

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

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

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

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

3.2 Matrices

Determinant.h

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

bd5cec, 15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs
        (a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *=
        -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -=
          v * a[i][k];
    }
  }
  return res;
}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}\left(N^3\right)$

3313dc, 18 lines

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

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}\left(n^2 m\right)$

44c9ab, 38 lines

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

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

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

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

SolveLinear2.h

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

"SolveLinear.h" 08e495, 7 lines

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

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}\left(n^2 m\right)$

fa2d7a, 34 lines

```
typedef bitset<1000> bs;

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

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

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank $<$ n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1})$ (mod p^k) where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}\left(n^3\right)$

ebfff6, 35 lines

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

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

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

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

3.3 Fourier transforms

FastFourierTransform.h

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

00ced6, 35 lines

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

NumberTheoreticTransform.h

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

"../number-theory/ModPow.h" ced03d, 33 lines

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

FastSubsetTransform.h

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

464cf3, 16 lines

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

Number theory (4)

4.1 Modular arithmetic

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)` = $\sum_{i=0}^{\mathrm{to}-1}(ki+c)\%m$. `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 $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.
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;
  }
}
```

ModFactorial.h

Description: Compute n! modulo p for small prime p.
Time: $\mathcal{O}\left(p + \log_p(n)\right)$

b6a9ad, 11 lines

```
int factmod(int n, int p) {
    vi f(p); f[0] = 1;
    rep(i, 1, p) f[i] = f[i-1]*i % p;
    int res = 1;
    while(n > 1) {
        if((n/p)%2) res = p - res;
        res = res * f[n%p] % p;
        n /= p;
    }
    return res;
}
```

4.2 Primality

FastEratosthenes.h

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

6b2912, 20 lines

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

MillerRabin.h

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

"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;
}
```

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 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)
$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$
$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$
Some standard results:
Euler Phi Function, $\phi(n) = \sum_{k|l} k\mu(\frac{l}{k})$
Number of co-prime pairs of integers (x, y) in the range $[1, n]$ = $\sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor^2$
Sum of $gcd(x, y)$ for all pairs in the range $[1, n]$ = $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor^2 \phi(i)$
Sum of $lcm(x, y)$ for all pairs in the range $[1, n]$ = $\sum_{i=1}^{n} (\frac{(1+\lfloor \frac{n}{i} \rfloor)(\lfloor \frac{n}{i} \rfloor)}{2})^2 \phi(i)$
Sum of $lcm(A[i], A[j])$ for all pairs of values in an array = $\sum_t \phi(t) (\sum_{a \epsilon A, t|a} a)^2$

Combinatorial (5)

5.1 Permutations

5.1.1 Factorial

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;
}
```

5.2 Partitions and subsets

5.2.1 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 $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

5.3 General purpose numbers

5.3.1 Eulerian numbers

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

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

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

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

5.3.2 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: $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$

Graph (6)

6.1 Fundamentals

BellmanFord.h

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

830a8f, 23 lines

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

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

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

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] = \texttt{inf}$ 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: $\mathcal{O}\left(N^3\right)$

531245, 12 lines

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

6.2 Network flow

Dinic.h

Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max|\text{cap}|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.

d7f0f1, 42 lines

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL);
        } // if you need flows
  };
  vi lvl, ptr, q;
  vector<vector<Edge>> adj;
  Dinic(int n) : lvl(n), ptr(n), q(n),
      adj(n) {}
  void addEdge(int a, int b, ll c, ll
      rcap = 0) {
    adj[a].push_back({b, sz(adj[b]), c, c
        });
    adj[b].push_back({a, sz(adj[a]) - 1,
        rcap, rcap});
  }
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < sz(adj[v]);
         i++) {
      Edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.
            c))) {
          e.c -= p, adj[e.to][e.rev].c +=
              p;
          return p;
        }
    }
    return 0;
  }
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    rep(L,0,31) do { // 'int L=30' maybe
        faster for random data
      lvl = ptr = vi(sz(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 -
              L))
            q[qe++] = e.to, lvl[e.to] =
                lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX))
          flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a
      ] != 0; }
};
```

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: $\mathcal{O}\left(V^3\right)$

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. Requires $N \leq M$.
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
}
```

6.4 DFS algorithms

CutVerticesAndEdges.h

Description: Finding cut vertices and cut edges.

969555, 48 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; // CUT VERTICE
    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) // CUT VERTICE
              IS_CUTPOINT(v);
            ++children;

            if (low[to] > tin[v]) // CUT
              EDGE
              IS_BRIDGE(v, to);
        }
    }
    if(p == -1 && children > 1) // CUT
      VERTICE
      IS_CUTPOINT(v);
}
void find_cutpoints() { // CUT VERTICE
    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);
    }
}
void find_bridges() { // CUT EDGE
    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);
    }
}
```

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Time: $\mathcal{O}(E+V)$

76b5c9, 24 lines

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

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

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;
  }
};
```

EulerWalk.h

Description: Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns an empty list if no cycle/path exists. Otherwise, returns a list of pairs node visited, edge index used. For the first nodes, edge index is always -1.
Time: $\mathcal{O}(V+E)$

413751, 17 lines

```
vector<pii> eulerWalk(vector<vector<pii>>
    &gr, int nedges, int src = 0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges);
  vector<pii> ret, s = {{src, -1}};
  D[src]++;
  while(!s.empty()) {
    auto [x, ind] = s.back();
    int y, e, &it = its[x], end = sz(gr[x
        ]);
    if(it == end) { ret.push_back({x, ind
        }); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if(!eu[e]) {
      D[x]--; D[y]++;
      eu[e] = 1; s.push_back({y, e});
    }}
  for(int x : D) if(x < 0 || sz(ret) !=
      nedges + 1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

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

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

b0d5b1, 12 lines

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

MaximumIndependentSet.h

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

6.7 Trees

BinaryLifting.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

3b98be, 32 lines

```
struct LCA {
  int n, l, timer;
  vector<vector<int>> adj, up;
  vector <int> tin, tout;
  LCA(vector<vector<int>> _adj, int root
     = 0) : adj(_adj) {
    n = adj.size();
    l = ceil(log2(n)) + 1;
    timer = 0;
    tin.resize(n);
    tout.resize(n);
    up.assign(n, vector<int>(l));
    dfs(root, root, 0);
  }
  void dfs(int u, int p, int d) {
    tin[u] = ++timer;
    up[u][0] = p;
    for(int i = 1; i < l; i++) { up[u][i]
       = up[up[u][i - 1]][i - 1]; }
    for(auto v : adj[u]) { if(v != p) dfs
       (v, u, d + 1); }
    tout[u] = ++timer;
  }
  bool is_ancestor(int x, int y) { return
      tin[x] <= tin[y] && tout[x] >=
     tout[y]; }
  int query(int u, int v) {
    if(is_ancestor(u, v)) return u;
    if(is_ancestor(v, u)) return v;
    for(int i = l - 1; i >= 0; i--) { if
       (!is_ancestor(up[u][i], v)) u =
       up[u][i]; }
    return up[u][0];
  }
  int jump(int x, int k) {
    for(int i = l - 1; i >= 0; i--) if((k
       >>i) & 1) x = up[x][i];
    return x;
  }
};
```

LCA.h

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

"../data-structures/RMQ.h" 0f62fb, 21 lines

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

  LCA(vector<vi>& C) : time(sz(C)), rmq((
     dfs(C,0,-1), ret)) {}
  void dfs(vector<vi>& C, int v, int par)
      {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
      path.push_back(v), ret.push_back(
         time[v]);
      dfs(C, y, v);
    }
  }

  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
  }
  //dist(a,b){return depth[a] + depth[b]
     - 2*depth[lca(a,b)];}
};
```

CompressTree.h

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

"LCA.h" 9775a0, 21 lines

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

Centroid.h

Description: Find centroid decomposition of a tree. Main function to be called is solve(root, root). Process trees formed after decomposition at each step within process(node).

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

0f3b3f, 23 lines

```
vector<vi> adj; vi siz;
vector<bool> vis;
void find_size(int v, int p) {
  siz[v] = 1;
  for(auto nx : adj[v]) if(nx != p && !
      vis[nx]) {
    find_size(nx, v);
    siz[v] += siz[nx];
  }
}
int centroid(int v, int p, int n) {
  for(auto nx : adj[v]) if(nx != p && !
      vis[nx] && siz[nx] > n/2) {
    return centroid(nx, v, n);
  }
  return v;
}
void process(int v) {}
void solve(int v, int p) {
  find_size(v, p);
  int c = centroid(v, p, siz[v]);
  process(c);
  vis[c] = true;
  for(auto nx : adj[c]) if(!vis[nx])
      solve(nx, c);
}
```

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 can support any sort of 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/LazySegmentTree.h" 1dd3c5, 46 lines

```
template <bool VALS_EDGES> struct HLD {
  int N, tim = 0;
  vector<vi> adj;
  vi par, siz, depth, rt, pos;
  LazySegTree<int, int> st;
  HLD(vector<vi> adj_)
    : N(sz(adj_)), adj(adj_), par(N, -1),
       siz(N, 1), depth(N),
      rt(N),pos(N),st(4*N) { dfsSz(0);
          dfsHld(0); }
  void dfsSz(int v) {
    if (par[v] != -1) adj[v].erase(find(
        all(adj[v]), par[v]));
    for (int& u : adj[v]) {
      par[u] = v, depth[u] = depth[v] +
          1;
      dfsSz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u
          , adj[v][0]);
    }
  }
  void dfsHld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u
          );
      dfsHld(u);
    }
  }
  template <class B> void process(int u,
      int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v
        ]]) {
      if (depth[rt[u]] > depth[rt[v]])
          swap(u, v);
      op(pos[rt[v]], pos[v]);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u] + VALS_EDGES, pos[v]);
  }
  void modifyPath(int u, int v, int val)
      {
    process(u, v, [&](int l, int r) { st.
        apply(l, r, val); });
  }
  int queryPath(int u, int v) {
    int res = st.SID;
    process(u, v, [&](int l, int r) {
        res = st.comb(res, st.query(l, r)
            );
    });
    return res;
  }
  int querySubtree(int v) { //
      modifySubtree is similar
    return st.query(pos[v] + VALS_EDGES,
        pos[v] + siz[v] - 1);
  }
};
```

Centroid.h

Description: Find centroid decomposition of a tree. Main function to be called is solve(root, root). Process trees formed after decomposition at each step within process(node).
Time: $\mathcal{O}(N \log N)$

0f3b3f, 23 lines

```
vector<vi> adj; vi siz;
vector<bool> vis;
void find_size(int v, int p) {
  siz[v] = 1;
  for(auto nx : adj[v]) if(nx != p && !
     vis[nx]) {
    find_size(nx, v);
    siz[v] += siz[nx];
  }
}
int centroid(int v, int p, int n) {
  for(auto nx : adj[v]) if(nx != p && !
     vis[nx] && siz[nx] > n/2) {
    return centroid(nx, v, n);
  }
  return v;
}
void process(int v) {}
void solve(int v, int p) {
  find_size(v, p);
  int c = centroid(v, p, siz[v]);
  process(c);
  vis[c] = true;
  for(auto nx : adj[c]) if(!vis[nx])
     solve(nx, c);
}
```

6.8 Extra

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;
}
```

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}(3^{n/3})$, much faster for sparse graphs

b0d5b1, 12 lines

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

MaximumIndependentSet.h

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

Strings (7)

KMP.h

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.
Time: $\mathcal{O}(n)$

d4375c, 16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}

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

Zfunc.h

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

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

ee09e2, 12 lines

```
vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i -
        l]);
    while (i + z[i] < sz(S) && S[i + z[i
        ]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

Manacher.h

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

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

e7ad79, 13 lines

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

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

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

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

d07a42, 8 lines

```
int minRotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(k,0,N) {
    if (a+k == b || s[a+k] < s[b+k]) {b
        += max(0, k-1); break;}
    if (s[a+k] > s[b+k]) { a = b; break;
        }
  }
  return a;
}
```

SuffixArray.h

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

2d2a67, 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.
typedef uint64_t ull;
struct H {
  ull x; H(ull x=0) : x(x) {}
  H operator+(H o) { return x + o.x + (x
      + o.x < x); }
  H operator-(H o) { return *this + ~o.x;
       }
  H operator*(H o) { auto m = (
      __uint128_t)x * o.x;
    return H((ull)m) + (ull)(m >> 64); }
  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;}
```

Various (8)

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);
}
```

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) > \ldots > f(b)$. If integer coordinates, use r-l $>$ 3. Change $<$ to $>$ if minimizing.
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$

55aee5, 11 lines

```
template <class F>
double ternary(double l, double r, F f) {
  assert(l <= r);
  while(r - l > eps) {
    double m1 = l + (r - l)/3;
    double m2 = r + (r - l)/3;
    if(f(m1) < f(m2)) l = m1;
    else r = m2;
  }
  return f(l);
}
```

FastKnapsack.h

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

b20ccc, 16 lines

```
int knapsack(vi w, int t) {
  int a = 0, b = 0, x;
  while (b < sz(w) && a + w[b] <= t) a +=
       w[b++];
  if (b == sz(w)) return a;
  int m = *max_element(all(w));
  vi u, v(2*m, -1);
  v[a+m-t] = b;
  rep(i,b,sz(w)) {
    u = v;
    rep(x,0,m) v[x+w[i]] = max(v[x+w[i]],
        u[x]);
    for (x = 2*m; --x > m;) rep(j, max(0,
       u[x]), v[x])
      v[x-w[j]] = max(v[x-w[j]], j);
  }
  for (a = t; v[a+m-t] < 0; a--) ;
  return a;
}
```

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

8.1 Optimization tricks

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

8.1.1 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).

RNG.h

Description: Fast random number generator.
Usage: `rng(); shuffle(all(v), rng);`

c8a385, 1 lines

```
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
```

EnumerateBits.h

Description: Enumerating all submasks of a given mask. (Including 0)

2c8302, 6 lines

```
// Iterating over all masks
for (int m=0; m<(1<<n); ++m)
    // Iterating all submasks of a given
       mask
    for (int s=m; s; s=(s-1)&m)
        ... you can use s ...
        if (s==0)  break;
```

FixedNumberOfSetBits.h

Description: Enumerating all masks with k set bits and smaller that MXVAL.

47074a, 13 lines

```
int MXVAL; //max allowed value
unsigned int v = (1ll<<k)-1; //Smallest
    mask
unsigned int mx = v;
while (mx < MXVAL)
    mx <<= 1;
mx >>= 1; //Greatest mask
while (true) {
    // ................
    if (v == mx)
        break;
    unsigned int t = v | (v - 1);
    v = (t + 1) | (((~t & -~t) - 1) >> (
       __builtin_ctz(v) + 1));
}
```

BitHacks.h

Description: Bit Hacks.

d41d8c, 10 lines

```
// x & -x@ is the least bit in \texttt{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.

// 1.) __builtin_popcount(x)
// 2.) __builtin_parity(x)
// 3.) __builtin_clz(x): Counts the
    leading number of zeros of the
    integer(long/long long).
// 4.) __builtin_ctz(x): Counts the
    trailing number of zeros of the
    integer(long/long long).
```

SOS.h

Description: Sum over subsets DP.

505a8d, 7 lines

```
for(int i = 0; i<(1<<N); ++i)
  F[i] = A[i];
for(int i = 0;i < N; ++i)
    for(int mask = 0; mask < (1<<N); ++
       mask){
      if(mask & (1<<i))
        F[mask] += F[mask^(1<<i)];
    }
```

Basis.h

Description: Finding the basis of a vector space with n vectors and d dimensions

e54101, 13 lines

```
int basis[d]; // basis[i] keeps the mask
   of the vector whose f value is i
int sz; // Current size of the basis
void insertVector(int mask) {
  for (int i = 0; i < d; i++) {
    if ((mask & 1 << i) == 0) continue;
       // continue if i != f(mask)
    if (!basis[i]) { // If there is no basis vector with the i'th bit set, then insert this vector into the basis
      basis[i] = mask;
      ++sz;
      return;
    }
    mask ^= basis[i]; // Otherwise subtract the basis vector from this vector
  }
}
```

Geometry (9)

9.1 Geometric primitives

Point.h

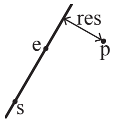
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

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> where T is e.g. long long or double.

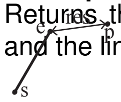


"Point.h" f6bf6b, 4 lines

```
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.



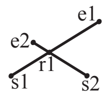
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"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

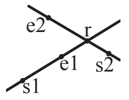
Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line 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 ⇔ 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);
}
```

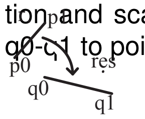
OnSegment.h

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();
}
```

9.2 Circles

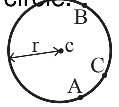
CircleIntersection.h

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

"Point.h" 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;
}
```

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};
}
```

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" 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

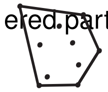
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"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;
}
```

ConvexHull.h

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



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

"Point.h" 310954, 13 lines

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

HullDiameter.h

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

"Point.h" c571b8, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S
      [0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2
          (), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S
          [i + 1] - S[i]) >= 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" 71446b, 14 lines

```
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 (sz(l) < 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;
}
```

9.4 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;
}
```

Junk.h

Description: Bakchodi

310d0b, 15 lines

```
|---------------------------------------|
|             _,--"^^"-.,_              |
|       _.-"~^`~-.     .-~`^~"-._       |
|   ,="`"-._     .----.      _.-"`"=,   |
|  ;_       "-. (0 )( 0)  .-"       _;  |
| .' `~"=,_    '.\ \/ /.'     _,="~` `. |
| ;_       "-. _.-)   (-._ .-"       _; |
| : ^~"-.,___.'  (    )  `.___,.-"~^ ;  |
| :        _:     `__'      :_        : |
|  '._,-~"` :':           :': `"~-,_.'  |
|     '.,_.-`.             .'`-._,.'    |
|        :__.-`-._______.-'`-.__;       |
|               //     \\               |
|             (((~     ~)))             |
|---------------------------------------|
```