IIIT Hyderabad

FLogic

Amul Agrawal, Aditya Verma, Alapan Chaudhuri

46th ICPC World Finals

April 2024

Contest (1)

template.cpp

19 lines

```cpp
#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()
#define pb push_back
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

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

// alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14
      \
  -fsanitize=undefined,address'
```

Data structures (2)

OrderedSet.h

28 lines

```cpp
/**
 * Author: Simon Lindholm
 * Date: 2016-03-22
 * License: CC0
 * Source: hacKIT, NWERC 2015
 * 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 \texttt{null\_type}.
 * Time: O(\log N)
 */
#pragma once

#include <bits/extc++.h> /** keep-include */
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
}
```

Dsu.h

29 lines

```cpp
/**
 * Author: Amul Agrawal
 * Date: 2022-09-18
 * Description: DSU with rollback
 * Time: $O(\alpha(N))$
 */
#pragma once

struct DSU {
  int sets; vi p, s;
  stack<pii> ss, sp;
  DSU(int n) : p(n, -1), s(n, 1), sets(n) {}
  bool IsSameSet(int a, int b) { return find(a) == find(b); }
  int find(int x) {return p[x] == -1 ? x : p[x] = find(p[x]);}
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (s[a] < s[b]) swap(a, b);
    ss.push({a, s[a]}); sp.push({b, p[b]});
    sets--; s[a] += s[b]; p[b] = a; return true;
  }
  int time() {return sz(ss);}
  void rollback(int t) {
    while(time() > t) {
      p[sp.top().first] = sp.top().second; sp.pop();
      s[ss.top().first] = ss.top().second; ss.pop();
    }
  }
};
```

DsuBp.h

26 lines

```cpp
/**
 * Author: Amul Agrawal
 * Date: 2022-09-18
 * Description: Graph, adding edges, checking bp color
 * Time: $O(\alpha(N))$
 */
#pragma once

struct DSU {
  int sets; vi p, s, l;
  DSU(int n) : p(n, -1), s(n, 1), l(n, 0), sets(n) {}
  bool IsSameColor(int a,int b) {
    find(a); find(b); return l[a] == l[b];
  }
  bool IsSameSet(int a, int b) { return find(a) == find(b); }
  int find(int x) {
    if(p[x] == -1) return x;
    int y = find(p[x]); l[x] ^= l[p[x]]; return p[x] = y;
  }
  void join(int a, int b) {
    int ca = a, cb = b; a = find(a), b = find(b);
    if (a == b) return;
    if (s[a] < s[b]) swap(a, b);
    sets--; s[a] += s[b]; l[b] = 1 ^ l[ca] ^ l[cb]; p[b] = a;
  }
};
```

MinQueue.h

32 lines

```cpp
/**
 * Author: Amul Agrawal
 * Date: 2022-09-18
 * Description: Minimum Queue Applications
 * Time: $O(1)$ push pop etc.
 */
#pragma once

template<class T>
struct MinQueue {
  deque<pair<T, T>> q;
  int ca = 0, cr = 0, plus = 0, sze = 0;
  void push(T x) {
    x -= plus;
    // change '>' to '<' and you get max-queue
    while (!q.empty() && q.back().first > x)
      q.pop_back();
    q.push_back({x, ca}); ca++; sze++;
  }
  T pop() {
    T re = 0;
    if (!q.empty() && q.front().second == cr) {
      re = q.front().first; q.pop_front();
    }
    cr++; sze--; return re + plus;
  }
  // Returns minimum in the queue
  T min() { return q.front().first + plus; }
  int size() { return sze; }
  // Adds x to every element in the queue
  void add(int x) { plus += x; }
};
```

Matrix.h

40 lines

```cpp
/**
 * Author: Ulf Lundstrom
 * Date: 2009-08-03
 * License: CC0
 * Source: My head
 * Description: Basic operations on square matrices.
 * Usage: Matrix<int, 3> A;
 *  A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
 *  vector<int> vec = {1,2,3};
 *  vec = (A^N) * vec;
 * Status: tested
 */
#pragma once

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

SparseTable.h

36 lines

```cpp
/**
 * Author: Johan Sannemo, pajenegod
 * Date: 2015-02-06
 * License: CC0
 * Source: Folklore
 * 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: $O(|V| \log |V| + Q)$
 * Status: stress-tested
 */
#pragma once

template<class T>
struct SparseTable {
    T (*op)(T, T);
    vi log2s; vector<vector<T>> st;
    SparseTable (const vector<T>& arr, T (*op)(T, T))
      : op(op), log2s(sz(arr)+1), st(sz(arr)) {
        rep(i,2,sz(log2s)) { log2s[i] = log2s[i/2] + 1; }
        rep(i,0,sz(arr)) {
          st[i].assign(log2s[sz(arr) - i] + 1);
          st[i][0] = arr[i];
        }
        rep(p, 1, log2s[sz(arr)] + 1) rep(i,0,sz(arr))
          if(i+(1<<p) <= sz(arr)) {
            st[i][p] = op(st[i][p-1], st[i+(1<<(p-1))][p-1]);
        }
    }
    T query (int l, int r) {
        int p = log2s[r-l+1];
        return op(st[l][p], st[r-(1<<p)+1][p]);
    }
};
```

FenwickTree.h — 34 lines

```
/**
 * Author: Lukas Polacek
 * Date: 2009-10-30
 * License: CC0
 * Source: folklore/TopCoder
 * Description: Computes partial sums a[0] + a[1] + ... + a[pos
       - 1], and updates single elements a[i],
 * taking the difference between the old and new value.
 * Time: Both operations are $O(\log N)$.
 * Status: Stress-tested
 */
#pragma once

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

FenwickTree2D.h — 36 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2017-05-11
 * License: CC0
 * Source: folklore
 * Description: Computes sums a[i,j] for all i<I, j<J, and
       increases single elements a[i,j].
 *  Requires that the elements to be updated are known in
       advance (call fakeUpdate() before init()).
 * Time: $O(\log^2 N)$. (Use persistent segment trees for $O(\
       log N)$.)
 * Status: stress-tested
 */
#pragma once

#include "FenwickTree.h"

struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
  }
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

SegmentTree.h — 49 lines

```
/**
 * Author: Amul Agrawal
 * Date: 2022-09-18
 * Description: RMQ SegTree
 * Time: $O(\log(N))$
 */
#pragma once
const ll INF = 1e18;
struct node {
  ll x;
};

template<class T>
struct SegmentTrees {
  vector<node> st, lazy;
  node def;
  SegmentTrees(int n) : st(4*n, {INF}), lazy(4*n, {INF}), def({
       INF}) {}
  inline node combine(node a, node b) {
    node ret; ret.x = min(a.x, b.x); return ret;
  }
  void push(int pos) {
    if(lazy[pos].x != INF) {
      st[pos*2] = lazy[pos]; st[pos*2 + 1] = lazy[pos];
      lazy[pos*2] = lazy[pos]; lazy[pos*2+1] = lazy[pos];
      lazy[pos] = def;
    }
  }
  void update(int l,int r,T val,int left,int right,int pos=1) {
    if(l > r) return;
    if(l==left && r==right) {
      st[pos].x = val; lazy[pos] = {val};
    } else {
      push(pos);
      int mid = (left + right)/2;
      update(l, min(r,mid), val, left, mid, pos*2);
      update(max(l,mid+1), r, val, mid+1, right, pos*2+1);
      st[pos] = combine(st[pos*2], st[pos*2+1]);
    }
  }
  node query(int l,int r,int left,int right,int pos=1) {
    if(l>r) return def;
    if(l==left && r==right) return st[pos];
    else {
      push(pos); int mid = (left + right)/2;
      return combine(query(l, min(r,mid), left, mid, pos*2),
        query(max(l,mid+1), r, mid+1, right, pos*2+1));
    }
  }
};
```

Treap.h — 176 lines

```
/**
 * Author: Amul Agrawal
 * Date: 2022-09-18
 * Description: cutting and moving array. everything is [l, r]
 *  0 based indexing.
 * Time: O(\log N)
 * Usage: Treap<int> tr(arr);
 */
#pragma once

struct node {
  int prior, val, min1, lazy, size;
  bool rev;
  node *l, *r;
};
typedef node* pnode;

template<class T = int>
class Treap {
public:
  pnode root;
  pnode getnode(T val) {
    pnode t = new node;
    t->l = t->r = NULL;
    t->prior = rand(); t->size = 1; t->rev = false;
    t->lazy = 0; t->min1 = t->val = val;
    return t;
  }
  inline int sz(pnode t) { return t ? t->size : 0;}
  // t may denote same node as l or r, so take care of that.
  void combine(pnode &t,pnode l,pnode r) {
    if(!l or !r) return void(t = (l ? l : r));
    t->size = sz(l) + sz(r); t->min1 = min(l->min1, r->min1);
  }
  void operation(pnode t) {
    if(!t) return;
    // reset t;
    t->size = 1; t->min1 = t->val;
    push(t->l); push(t->r);
    // combine
    combine(t, t->l, t); combine(t, t, t->r);
```

```
  }
  void push(pnode t) {
    if(!t) return;
    if(t->rev) {
      swap(t->r, t->l);
      if(t->r) t->r->rev = not t->r->rev;
      if(t->l) t->l->rev = not t->l->rev;
      t->rev = false;
    }
    if(t->lazy) {
      t->val += t->lazy;
      t->min1 += t->lazy;
      if(t->r) t->r->lazy += t->lazy;
      if(t->l) t->l->lazy += t->lazy;
      t->lazy = 0;
    }
  }
  // l = [0, pos], r = rest
  void split(pnode t,pnode &l,pnode &r,int pos,int add=0) {
    push(t);
    if(!t) return void(l=r=NULL);
    int curr_pos = add + sz(t->l);
    if(pos >= curr_pos) {
      split(t->r,t->r, r, pos, curr_pos + 1);
      l = t;
    } else {
      split(t->l, l, t->l, pos, add);
      r = t;
    }
    operation(t);
  }
  void merge(pnode &t,pnode l,pnode r) {
    push(l); push(r);
    if(!l or !r) return void(t = (l ? l : r));
    if(l->prior > r->prior) {
      merge(l->r, l->r, r);
      t = l;
    } else {
      merge(r->l, l, r->l);
      t = r;
    }
    operation(t);
  }
  void heapify(pnode t) {
    if(!t) return ;
      pnode max = t;
      if (t->l != NULL && t->l->prior > max->prior)
          max = t->l;
      if (t->r != NULL && t->r->prior > max->prior)
          max = t->r;
      if (max != t) {
          swap (t->prior, max->prior);
          heapify (max);
      }
  }
  // O(n) treap build given array is increasing
  pnode build(T *arr,int n) {
    if(n==0) return NULL;
    int mid = n/2;
    pnode t = getnode(arr[mid]);
    t->l = build(arr, mid);
    t->r = build(arr + mid + 1, n - mid - 1);
    heapify(t); operation(t);
    return t;
  }
  Treap(vector<T> &arr) {
    root = NULL;
    for(int i=0;i<arr.size();i++) {
      T c = arr[i];
      merge(root, root, getnode(c));
    }
  }
  void add(int l,int r,T d) {
    if(l>r) return;
    pnode L, mid, R;
    split(root, L, mid, l-1); split(mid, mid, R, r-l);
    if(mid) {
      mid->lazy += d;
    }
    merge(L, L, mid); merge(root, L, R);
  }
  void reverse(int l,int r) {
    if(l>r) return;
    pnode L, mid, R;
    split(root, L, mid, l-1); split(mid, mid, R, r-l);
    if(mid) {
      mid->rev = not mid->rev;
    }
    merge(R, mid, R); merge(root, L, R);
  }
  void revolve(int l,int r,int cnt) {
    if(cnt<=0 or l>r) return;
    int len = r - l + 1;
    // cnt = len => no rotation;
    cnt %= len;
    if(cnt == 0) return;
    // pick cnt elements from the end  // => (len - cnt) from
        front
    int mid = l + (len - cnt) - 1; pnode L, Range, R;
    split(root, L, Range, l-1); split(Range, Range, R, r - l);
    pnode first, second;
    split(Range, first, second, (len-cnt-1));
    merge(Range, second, first);
    merge(L, L, Range); merge(root, L, R);
  }
  void insert(int after,T val) {
    pnode L, R; split(root, L, R, after);
    merge(L, L, getnode(val)); merge(root, L, R);
  }
  void del(int pos) {
    pnode L, mid, R;
    split(root, L, mid, pos-1); split(mid, mid, R, 0);
    if(mid) {
      delete mid;
    }
    merge(root, L, R);
  }
  T range_min(int l,int r) {
    pnode L, mid, R;
    split(root, L, mid, l-1); split(mid, mid, R, r-l);
    push(mid); T ans = mid->min1;
    merge(L, L, mid); merge(root, L, R);
    return ans;
  }
  void inorder(pnode curr) {
    push(curr); if(!curr) return;
    inorder(curr->l); cerr<<curr->val<<" "; inorder(curr->r);
  }
  int query(int pos) {
    pnode l, mid, r;
    split(root, l, mid, pos-1); split(mid, mid, r, 0);
    int ans = mid->val;
    merge(l, l, mid); merge(root, l, r);
    return ans;
  }
};
```

SQRT.h — 73 lines

```
/**
 * Author: Amul Agrawal
 * Date: 2022-09-18
 * Description: Square Root Decomposition
 * Time: Amul Knows
 */
#pragma once

const int N = 1e5 + 13, Q = 1e5 + 13, B = 500;
int S[N/B + 13][B + 13], len[N/B + 13], prv[N], nxt[N], st[N/B
    + 13], en[N/B + 13], A[N];
map<int,set<int>> pos; int n, q;

void add_link(int p,int val) {
    nxt[p] = val; prv[val] = p;
    if(p < 1 or p > n) return;
    int b = p / B;
    for(int i = st[b]; i <= en[b]; i++) {
        S[b][i - st[b] + 1] = nxt[i];
    }
    sort(S[b] + 1, S[b] + len[b] + 1);
}
// set A_x = y
void point_update(int x,int y) {
    // update the original link
    add_link(prv[x], nxt[x]); pos[A[x]].erase(x);
    // insert new links
    A[x] = y; pos[A[x]].insert(x);
    int pr = 0, nx = n + 1;
    if(*pos[A[x]].begin() != x) pr = *prev(pos[A[x]].find(x));
    if(*pos[A[x]].rbegin() != x) nx = *next(pos[A[x]].find(x));
    add_link(pr, x); add_link(x, nx);
}

int query_block(int s,int e,int k) {
    int ans = 0;
    for(int i = s; i <= e; i++)
        ans += ((S[i] + len[i] + 1) - upper_bound(S[i] + 1, S[i
            ] + len[i] + 1, k));
    return ans;
}

int query_elements(int s,int e,int k) {
    int ans = 0;
    for(int i = s; i <= e; i++)
        ans += (nxt[i] > k);
    return ans;
}

int range_query(int l,int r) {
    int lb = l / B, rb = r / B;
    if(lb == rb) return query_elements(l, r, r);
    return query_elements(l, en[lb], r)
        + query_block(lb + 1, rb - 1, r)
        + query_elements(st[rb], r, r);
}
for(int i = 1; i <= n; i++) {
    nxt[i] = n + 1;
    if(!pos[A[i]].empty()) {
        prv[i] = *pos[A[i]].rbegin();
        nxt[prv[i]] = i;
    }
    pos[A[i]].insert(i);
}
for(int i = 1; i <= n; i++) {
    int b = i / B;
    if(!len[b])
```

```
        st[b] = i;
    en[b] = i;
    len[b]++;
    S[b][len[b]] = nxt[i];
}
for(int i = 0; i <= n/B; i++) {
    sort(S[i] + 1, S[i] + len[i] + 1);
}
```

LazyDynamicSegTree.h — 40 lines

```
/**
 * Author: Amul Agrawal
 * Date: 2022-09-18
 * Description: Segment Tree based on large [L, R] range
 *(includes range updates)
 * Time: $O(\log(R - L))$ in addition and deletion
 */
#pragma once

using T=ll; using U=ll;   // exclusive right bounds
T t_id; U u_id; // t_id: total (normal), u_id: lazy (default)
T op(T a, T b){ return a+b; }
void join(U &a, U b){ a+=b; }
void apply(T &t, U u, int x){ t+=x*u; }
T part(T t, int r, int p){ return t/r*p; }
struct DynamicSegmentTree {
  struct Node { int l, r, lc, rc; T t; U u;
    Node(int l, int r):l(l),r(r),lc(-1),rc(-1),t(t_id),u(u_id){
        }
  };
  vector<Node> tree;
  DynamicSegmentTree(int N) { tree.push_back({0,N}); }
  void push(Node &n, U u){ apply(n.t, u, n.r-n.l); join(n.u,u);
       }
  void push(Node &n){push(tree[n.lc],n.u);push(tree[n.rc],n.u);
      n.u=u_id;}
  T query(int l, int r, int i = 0) { auto &n = tree[i];
    if(r <= n.l || n.r <= l) return t_id;
    if(l <= n.l && n.r <= r) return n.t;
    if(n.lc < 0) return part(n.t, n.r-n.l, min(n.r,r)-max(n.l,l
        ));
    return push(n), op(query(l,r,n.lc),query(l,r,n.rc));
  }
  void update(int l, int r, U u, int i = 0) { auto &n = tree[i
      ];
    if(r <= n.l || n.r <= l) return;
    if(l <= n.l && n.r <= r) return push(n,u);
    if(n.lc < 0) { int m = (n.l + n.r) / 2;
      n.lc = tree.size();       n.rc = n.lc+1;
      tree.push_back({tree[i].l, m}); tree.push_back({m, tree[i
          ].r});
    }
    push(tree[i]); update(l,r,u,tree[i].lc); update(l,r,u,tree[
        i].rc);
    tree[i].t = op(tree[tree[i].lc].t, tree[tree[i].rc].t);
  }
};
```

LineContainer.h — 42 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * 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: O(\log N)
 * Status: stress-tested
 */
#pragma once

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;
  }
};
```

Graph (3)

Dinic.h — 76 lines

```
/**
 * Author: chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://cp-algorithms.com/graph/dinic.html
 * Description: Complexity: (1) O($V^2 E$): General
 * (2) O($\text{Flow} E$): General
 * (3) O($E \sqrt{V}$): when sum of edge capacities
 * is O($n$), we can treat edge with weight $x$ as $x$
 * edges with weight 1.
 * (4) O($E V \log(Flow)$): Dinics with scaling
 * Status: Tested on SPOJ FASTFLOW and SPOJ MATCHING, stress-
     tested
 */
#pragma once

const int INF = 1e9 + 13;
template<class T = long long>
class Dinic {
  // {to: to, rev: reverse_edge_id, c: cap, oc: original cap}
  struct Edge {
    int to, rev;
    T c, oc;
    T flow() { return max(oc - c, (T)0); } // if you need flows
  };
  int N;
  vector<int> lvl, ptr, q; vector<vector<Edge>> adj;
public:
  vector<vector<T>> Flow;
  Dinic(int n) {
    N = n; Flow.assign(n, vector<T>(n, (T)0));
    lvl.resize(n); adj.resize(n); ptr.resize(n); q.resize(n);
  }
  // automatically adds a reversed edge
  void addEdge(int a, int b, T c, T rcap = 0) {
    adj[a].push_back({b, sz(adj[b]), c, c});
    adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
  }
  T dfs(int v, int t, T 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 (T p = dfs(e.to, t, min(f, e.c))) {
          e.c -= p, adj[e.to][e.rev].c += p;
          return p;
        }
    }
    return 0;
  }
  T calc(int s, int t) {
    T flow = 0; q[0] = s;
    // bfs part, setting the lvl here
    for(int L = 0; L < 31; L++) do { // 'int L=30' maybe faster
        for random data
      lvl = ptr = vector<int>(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;
      }
      // dfs part, setting ptr and checking for a path.
      while (T p = dfs(s, t, INF)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
  void buildFlow() {
    for(int i=0;i<N;i++) {
      for(auto e : adj[i]) {
        int j = e.to;
        Flow[i][j] = e.flow();
      }
    }
  }
};
```

MinCut.h — 7 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2015-05-13
 * Source: Wikipedia
 * 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.
 * Status: works
 */
```

GlobalMinCut.h — 32 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2021-01-09
```

```
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Stoer%E2%80%93
     Wagner_algorithm
 * Description: Find a global minimum cut in an undirected
     graph, as represented by an adjacency matrix.
 * Time: O(V^3)
 * Status: Stress-tested together with GomoryHu
 */
#pragma once

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;
}
```

MinCostMaxFlow.h

93 lines

```
/**
 * Author: Stanford
 * Date: Unknown
 * Source: Stanford Notebook
 * 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.
 * Status: Tested on kattis:mincostmaxflow, stress-tested
     against another implementation
 * Time: Approximately O(E^2)
 */
#pragma once

// #include <bits/extc++.h> /// include-line, keep-include

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
  }
};
```

hopcroftKarp.h

62 lines

```
/**
 * Author: Chen Xing
 * Date: 2009-10-13
 * License: CC0
 * Source: N/A
 * 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: O(\sqrt{V}E)
 * Status: stress-tested by MinimumVertexCover, and tested on
     oldkattis.adkbipmatch and SPOJ:MATCHING
 */
#pragma once

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);
    /// Find the starting nodes for BFS (i.e. layer 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);
    /// Find all layers using bfs.
    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);
    }
    /// Use DFS to scan for augmenting paths.
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
  return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

32 lines

```
/**
 * Author: Johan Sannemo, Simon Lindholm
 * Date: 2016-12-15
 * License: CC0
```

```
 * 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.
 * Status: stress-tested
 */
#pragma once

#include "hopcroftKarp.h"
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = hopcroftKarp(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

47 lines

```
/**
 * Author: Benjamin Qi, chilli
 * Date: 2020-04-04
 * License: CC0
 * Source: https://github.com/bqi343/USACO/blob/master/
     Implementations/content/graphs%20(12)/Matching/Hungarian.
     h
 * Description: Given a weighted bipartite graph, matches every
      node on
 * the left with a node on the right such that no
 * nodes are in two matchings and the sum of the edge weights
     is minimal. Takes
 * cost[N][M], where cost[i][j] = cost for L[i] to be matched
     with R[j] and
 * returns (min cost, match), where L[i] is matched with
 * R[match[i]]. Negate costs for max cost.
 * Time: O(N^2M)
 * Status: Tested on kattis:cordonbleu, stress-tested
 */
#pragma once

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
}
```

2sat.h

75 lines

```
/**
 * Author: Emil Lenngren, Simon Lindholm
 * Date: 2011-11-29
 * License: CC0
 * Source: folklore
 * 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 (\texttt
     {\tilde{}x}).
 * Usage:
 *  TwoSat ts(number of boolean variables);
 *  ts.either(0, \tilde3); // Var 0 is true or var 3 is false
 *  ts.setValue(2); // Var 2 is true
 *  ts.atMostOne({0,\tilde1,2}); // <= 1 of vars 0, \tilde1 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: O(N+E), where N is the number of boolean variables,
     and E is the number of clauses.
 * Status: stress-tested
 */
#pragma once

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

31 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2019-12-31
 * License: CC0
 * Source: folklore
 * Description: Eulerian undirected/directed path/cycle
     algorithm.
 * Input should be a vector of (dest, global edge index), where
 * for undirected graphs, forward/backward edges have the same
     index.
 * Returns a list of nodes in the Eulerian path/cycle with src
     at both start and end, or
 * empty list if no cycle/path exists.
 * To get edge indices back, add .second to s and ret.
 * Time: O(V + E)
 * Status: stress-tested
 */
#pragma once

vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

CondensationGraph.h

70 lines

```
/**
 * Author: Lukas Polacek
 * Date: 2009-10-28
 * License: CC0
 * Source: Czech graph algorithms book, by Demel. (Tarjan's
     algorithm)
 * 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: O(E + V)
 * Status: Bruteforce-tested for N <= 5
 */
#pragma once

// 0 based indexing
void condense(vector<vi> adj,vector<vi> &adj_scc,
            vector<vi> &comp,vi &root_of,int n) {
  vector<vi> rev_adj(n);
  rep(u,0,n) {
    for(auto v : adj[u]) {
      rev_adj[v].push_back(u);
    }
  }
  vector<bool> vis(n, false); vi order, component, root_nodes;
  function<void(int)> dfs1 = [&](int x) {
    vis[x] = true;
    for(auto nx : adj[x]) {
      if(!vis[nx]) {
        dfs1(nx);
      }
    }
    order.push_back(x);
  };
  rep(i, 0, n) { if(!vis[i]) dfs1(i); }
  vis.clear(); vis.assign(n, false);
  // order is now kind of topologically sorted
  reverse(order.begin(), order.end());
  function<void(int)> dfs2 = [&](int x) {
    vis[x] = true;
    component.push_back(x);
    for(auto u : rev_adj[x]) {
      if(!vis[u]) {
        dfs2(u);
      }
    }
  };
  comp.clear(); comp.resize(n);
  root_of.clear(); root_of.resize(n);
  for(auto v : order) {
    if(!vis[v]) {
      dfs2(v);
      int root = component.front();
      for(auto u : component) root_of[u] = root;
      root_nodes.push_back(root);
      comp[root] = component;
      component.clear();
    }
  }
  adj_scc.clear(); adj_scc.resize(n);
  rep(u, 0, n) {
    for(auto v : adj[u]) {
      if(root_of[u] != root_of[v]) {
        adj_scc[root_of[u]].push_back(root_of[v]);
      }
    }
  }
}
```

BridgeTree.h

42 lines

```
/**
 * Author: Amul Agrawal
 * Date: 2022-09-23
 * 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.
 */
#pragma once

// 0 based indexing
int n, m, Tin;
vector<vii> adj, adjn;
vi vis, low;
vector<array<int, 3>> bridges;
Dsu<int> ds;

int dfs0(int x,int p=-1,int w=0) {
  vis[x] = 1; low[x] = Tin++;
  int crl = low[x];
  for(auto nx : adj[x]) {
    if(nx.ff == p) continue;
    else if (vis[nx.ff]) crl = min(crl, low[nx.ff]);
    else crl = min(crl, dfs0(nx.ff, x, nx.ss));
  }
  if(crl == low[x] and p != -1) bridges.pb({x, p, w});
  else if (p != -1) ds.join(x, p);
  return crl;
}

void build_bridgetree() {
  // CLEAR global variables
  ds.build(n); // INITIALIZE DSU HERE
  rep(i,0,n) if(!vis[i]) dfs0(i);
  for(auto arr : bridges) {
    int u = ds.find(arr[0]), v = ds.find(arr[1]), w = arr[2];
    if(u != v) {
      adjn[v].pb({u, w}); adjn[u].pb({v, w});
    }
  }
}
```

EdgeColoring.h

46 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2020-10-12
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Misra_%26
     _Gries_edge_coloring_algorithm
 * https://codeforces.com/blog/entry/75431 for the note about
     bipartite graphs.
 * 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: O(NM)
 * Status: stress-tested, tested on kattis:gamescheduling
 */
#pragma once

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

29 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2018-07-18
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Bron%E2%80%93
     Kerbosch_algorithm
 * 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: O(3^{n/3}), much faster for sparse graphs
 * Status: stress-tested
 */
#pragma once
/// Possible optimization: on the top-most
/// recursion level, ignore 'cands', and go through nodes in
     order of increasing
/// degree, where degrees go down as nodes are removed.
/// (mostly irrelevant given MaximumClique)

typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
  if (!P.any()) { if (!X.any()) f(R); return; }
```

```
  auto q = (P | X)._Find_first();
  auto cands = P & ~eds[q];
  rep(i,0,sz(eds)) if (cands[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

MaximumClique.h — 61 lines

```
/**
 * Author: chilli, SJTU, Janez Konc
 * Date: 2019-05-10
 * License: GPL3+
 * Source: https://en.wikipedia.org/wiki/
     MaxCliqueDyn_maximum_clique_algorithm, https://gitlab.com
     /janezkonc/mcqd/blob/master/mcqd.h
 * Description: Quickly finds a maximum clique of a graph (
     given as symmetric bitset
 * matrix; self-edges not allowed). Can be used to find a
     maximum independent
 * set by finding a clique of the complement graph.
 * Time: Runs in about 1s for n=155 and worst case random
     graphs (p=.90). Runs
 * faster for sparse graphs.
 * Status: stress-tested
 */
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k,mnk,mxk + 1) for (int i : C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() { init(V), expand(V); return qmax; }
  Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

MaximumIndependentSet.h — 7 lines

```
/**
 * Author: chilli
 * Date: 2019-05-17
 * Source: Wikipedia
 * Description: To obtain a maximum independent set of a graph,
      find a max
 * clique of the complement. If the graph is bipartite, see
     MinimumVertexCover.
 */
```

LinkCutTree.h — 102 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2016-07-25
 * Source: https://github.com/ngthanhtrung23/ACM_Notebook_new/
     blob/master/DataStructure/LinkCut.h
 * Description: Represents a forest of unrooted trees. You can
     add and remove
 * edges (as long as the result is still a forest), and check
     whether
 * two nodes are in the same tree.
 * Time: All operations take amortized O(\log N).
 * Status: Stress-tested a bit for N <= 20
 */
#pragma once

struct Node { // Splay tree. Root's pp contains tree's parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc. if wanted)
  }
  void pushFlip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this : -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      z->c[h ^ 1] = b ? x : this;
    }
    y->c[i ^ 1] = b ? this : x;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
  }
  void splay() { /// Splay this up to the root. Always finishes
       without flip set.
    for (pushFlip(); p; ) {
      if (p->p) p->p->pushFlip();
      p->pushFlip(); pushFlip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() { /// Return the min element of the subtree
       rooted at this, splayed to the top.
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};

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

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node* u) { /// Move u to root of represented
       tree.
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) { /// Move u to root aux tree. Return
       the root of the root aux tree.
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

DirectedMST.h — 75 lines

```
/**
 * Author: chilli, Takanori MAEHARA, Benq, Simon Lindholm
```

```
 * Date: 2019-05-10
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/
     master/graph/arborescence.cc
 * and https://github.com/bqi343/USACO/blob/42
     d177dfb9d6ce350389583cfa71484eb8ae614c/Implementations/
     content/graphs%20(12)/Advanced/DirectedMST.h for the
     reconstruction
 * Description: Finds a minimum spanning
 * tree/arborescence of a directed graph, given a root node. If
      no MST exists, returns -1.
 * Time: O(E \log V)
 * Status: Stress-tested, also tested on NWERC 2018
     fastestspeedrun
 */
#pragma once

#include "../data-structures/UnionFindRollback.h"

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

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

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

HLD.h

85 lines

```
/**
 * Author: Benjamin Qi, Oleksandr Kulkov, chilli
 * Date: 2020-01-12
 * Description: Heavy Light Decomposition
 * License: CC0
 * Source: Path updates and Query in a Tree. Get Segment Tree
     first.
 * Time: O(logN * Time taken by Range Query DS)
 * Status: stress-tested against old HLD
 */
#pragma once

// requires a segment tree with init function
class HLD {
    SegmentTrees sgt; vector<vi> adj;
    vi sz, par, head, sc, st, ed;
    int t, n;
public:
    HLD(vector<vector<int>> &adj1,int n1): sz(n1+1), par(n1+1),
       head(n1+1), sc(n1+1), st(n1+1), ed(n1+1) {
        n = n1; adj = adj1; t = 0;
    }
    void dfs_sz(int x,int p = 0) {
        sz[x] = 1; par[x] = p; head[x] = x;
        for(auto nx : adj[x]) {
            if(nx == p) continue;
            dfs_sz(nx, x);
            sz[x] += sz[nx];
            if(sz[nx] > sz[sc[x]]) sc[x] = nx;
        }
    }
    void dfs_hld(int x,int p = 0) {
        st[x] = t++;
        if(sc[x]) {
            head[sc[x]] = head[x];
            dfs_hld(sc[x], x);
        }
        for(auto nx : adj[x]) {
            if(nx == p or nx == sc[x]) continue;
            dfs_hld(nx, x);
        }
        ed[x] = t - 1;
    }
    void build(int base = 1) {
        dfs_sz(base);
        dfs_hld(base);
        sgt.init(t);
    }
    bool anc(int x,int y) {
        if(x == 0) return true; if(y == 0) return false;
        return (st[x] <= st[y] and ed[x] >= ed[y]);
    }
    int lca(int x,int y) {
        if(anc(x, y)) return x; if(anc(y, x)) return y;
        while(!anc(par[head[x]], y)) x = par[head[x]];
        while(!anc(par[head[y]], x)) y = par[head[y]];
        x = par[head[x]]; y = par[head[y]];
        // one will overshoot the lca and the other will reach
             lca.
        return anc(x, y) ? y : x;
    }
    void update_up(int x,int p,ll add) {
        while(head[x] != head[p]) {
            sgt.update(st[head[x]], st[x], add, 0, t-1);
            x = par[head[x]];
        }
        sgt.update(st[p], st[x], add, 0, t - 1);
    }
    void range_update(int u,int v,T add) {
        int l =  lca(u, v);
        update_up(u, l, add); update_up(v, l, add);
        update_up(l, l, -add);
    }
    T query_up(int x,int p) {
        T ans = 0;
        while(head[x] != head[p]) {
            ans = min(ans, sgt.query(st[head[x]], st[x], 0, t
                 -1));
            x = par[head[x]];
        }
        ans = min(ans, sgt.query(st[p], st[x], 0, t - 1));
        return ans;
    }
    T range_min(int u,int v) {
        int l = lca(u, v);
        return min(query_up(u, l), query_up(v, l));
    }
};
```

CentroidDecomposition.h

67 lines

```
/**
 * Author: Tanuj Khattar
 * Description:
 * Time: $O(N \log N + Q)$
 * Status: stress-tested
 */

const int N = 5e4 + 13, logN = 17;
vi adj[N], sub(N), par(N,-1), lvl(N), done(N), par_adj(N);
vector<vi> dist(N, vi(logN, 0)), anc(N, vi(logN, 0));
int nn = 0, root;
void dfs_size(int x,int p) {
  nn++; sub[x] = 1;
  for(auto nx : adj[x]) if(!done[nx] and nx != p) {
      dfs_size(nx, x); sub[x] += sub[nx];
    }
}
int find_ct(int x,int p) {
  for(auto nx : adj[x]) if(!done[nx] and nx != p and sub[nx] >
       nn/2)
    return find_ct(nx, x);
  return x;
}
void dfs(int x,int p,int ct) {
  anc[x][lvl[ct]] = ct;
  for(auto nx : adj[x]) if(!done[nx] and nx != p) {
      dist[nx][lvl[ct]] = 1 + dist[x][lvl[ct]];
      dfs(nx, x, ct);
    }
}
// par_adj[ct] = adjacent vertex to parent of ct in OT in
     subtree of ct.
int decompose(int x,int p=-1) {
  nn = 0; dfs_size(x, x);
```

```
  int ct = find_ct(x, x);
  if(p) lvl[ct] = 1 + lvl[p];
  done[ct] = 1; par[ct] = p;
  dfs(ct, ct, ct);
  for(auto nx : adj[ct]) if(!done[nx]) {
      int nct = decompose(nx, ct);
      par_adj[nct] = nx;
    }
  return ct;
}
vector<vi> child_cntb(N), my(N);
rep(x,0,n) for(int y = x; y >= 0; y = par[y]) {
      my[y].pb(dist[x][lvl[y]]);
      if(par[y] >= 0)
        child_cntb[y].pb(dist[x][lvl[par[y]]]);
  }
rep(x,0,n) {
    sort(all(my[x])); sort(all(child_cntb[x]));
}
// number of nodes <= k in v.
auto cnt_k = [&](vi &v,int k) {
  int l = upper_bound(all(v), k) - v.begin();
  return l;
};
auto k_dists = [&](int x,int k) {
  int ans = cnt_k(my[x], k);
  int ch = x, q = x; x = par[x];
  while(x >= 0) {
    ans += (cnt_k(my[x], k - dist[q][lvl[x]]));
    ans -= (cnt_k(child_cntb[ch], k - dist[q][lvl[x]]));
    ch = x; x = par[x];
  }
  return ans;
};
```

AuxiliaryTrees.h

```
/**
 * Author: Amul
 * Description: Creates a auxiliary tree of $k$ nodes.
 * Time: $O(k)$
 */

#pragma once

using vvi = vector<vector<int>>
struct Tree {
  int n;
  vvi adj;
  vi pos, tour, depth, pos_end, max_depth, dp, max_up;
  Tree(int n) : n(n), adj(n), max_depth(n), dp(n), max_up(n) {}
  void add_edge(int s, int t) {
    adj[s].pb(t); adj[t].pb(s);
  }
  vvi table;
  int argmin(int i, int j) { return depth[i] < depth[j] ? i : j
      ; }
  void rootify(int r) {
    pos.resize(n); pos_end.resize(n);
    function<void (int,int,int)> dfs = [&](int u, int p, int d)
        {
      pos[u] = pos_end[u] = depth.size();
      tour.pb(u); depth.pb(d);
      for (int v: adj[u]) {
        if (v != p) {
          dfs(v, u, d+1);
          pos_end[u] = depth.size();
          tour.pb(u);
          depth.pb(d);
        }
      }
    }; dfs(r, r, 0);
    int logn = sizeof(int)*__CHAR_BIT__-1-__builtin_clz(tour.
        size()); // log2
    table.resize(logn+1, vi(tour.size()));
    iota(all(table[0]), 0);
    for (int h = 0; h < logn; ++h)
      for (int i = 0; i+(1<<h) < tour.size(); ++i)
        table[h+1][i] = argmin(table[h][i], table[h][i+(1<<h)])
            ;
  }
  int lca(int u, int v) {
    int i = pos[u], j = pos[v]; if (i > j) swap(i, j);
    int h = sizeof(int)*__CHAR_BIT__-1-__builtin_clz(j-i); // =
        log2
    return i == j ? u : tour[argmin(table[h][i], table[h][j
        -(1<<h)])];
  }
  int getDepth(int u){
    return depth[pos[u]];
  }
  void aux_Tree(vi nodes, vvi & adj_aux, vi & start_times){
    // adj_aux stores the children
    for(int x : nodes) start_times.pb(pos[x]);
    sort(all(start_times));
    for(int i = 1; i < (int) nodes.size();  i++){
      start_times.pb(pos[lca(tour[start_times[i]], tour[
          start_times[i - 1]])]);
    }
    sort(all(start_times));
    start_times.erase(unique(start_times.begin(), start_times.
        end()), start_times.end());
    adj_aux.resize(start_times.size());
    stack<int> st;
        // nodes now indexed according to start_times
    st.push(0);
    for(int i = 1; i < (int)start_times.size(); i++){
      while(pos_end[tour[start_times[st.top()]]] < start_times[
          i]){
        st.pop();
      }
      adj_aux[st.top()].pb(i);
      st.push(i);
    }
  }
};
```

Blossom.h

```
/**
 * Author: Aditya
 * Description: Blossom Algorithm
 */

#pragma once

vector<int> Blossom(vector<vector<int>>& graph) {
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
              orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        } else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

Strings (4)

Hashing.h

```
/**
 * Author: Simon Lindholm
 * Date: 2015-03-15
 * License: CC0
 * Source: own work
 * Description: Various self-explanatory methods for string
      hashing.
 * Use on Codeforces, which lacks 64-bit support and where
      solutions can be hacked.
 * Status: stress-tested
 */
#pragma once

typedef uint64_t ull;
static int C; // initialized below

// Arithmetic mod two primes and 2^32 simultaneously.
// "typedef uint64_t H;" instead if Thue-Morse does not apply.
template<int M, class B>
struct A {
  int x; B b; A(int x=0) : x(x), b(x) {}
  A(int x, B b) : x(x), b(b) {}
  A operator+(A o){int y = x+o.x; return{y - (y>=M)*M, b+o.b};}
```

```
  A operator-(A o){int y = x-o.x; return{y + (y< 0)*M, b-o.b};}
  A operator*(A o) { return {(int)(1LL*x*o.x % M), b*o.b}; }
  explicit operator ull() { return x ^ (ull) b << 21; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;

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];
  }
};

#include <sys/time.h>
int main() {
  timeval tp;
  gettimeofday(&tp, 0);
  C = (int)tp.tv_usec; // (less than modulo)
  assert((ull)(H(1)*2+1-3) == 0);
  // ...
}
```

Kmp.h

27 lines

```
/**
 * Author: Johan Sannemo
 * Date: 2016-12-15
 * License: CC0
 * 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: O(n)
 * Status: Tested on kattis:stringmatching
 */
#pragma once

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;
}
```

Manacher.h

23 lines

```
/**
 * Author: User adamant on CodeForces
 * Source: http://codeforces.com/blog/entry/12143
 * 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: O(N)
 * Status: Stress-tested
 */
#pragma once

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

20 lines

```
/**
 * Author: Stjepan Glavina
 * License: Unlicense
 * Source: https://github.com/stjepang/snippets/blob/master/
     min_rotation.cpp
 * Description: Finds the lexicographically smallest rotation
     of a string.
 * Time: O(N)
 * Usage:
 *  rotate(v.begin(), v.begin()+minRotation(v), v.end());
 * Status: Stress-tested
 */
#pragma once

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

42 lines

```
/**
 * Author: chilli
 * Date: 2019-04-11
 * License: Unknown
 * Source: Suffix array - a powerful tool for dealing with
     strings
 * (Chinese IOI National team training paper, 2009)
 * Description: Builds suffix array for a string.
 * \texttt{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 \texttt{sa[0] = n}
     .
 * The \texttt{lcp} array contains longest common prefixes for
 * neighbouring strings in the suffix array:
 * \texttt{lcp[i] = lcp(sa[i], sa[i-1])}, \texttt{lcp[0] = 0}.
 * The input string must not contain any zero bytes.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once

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++);
  }
};
```

Z.h

21 lines

```
/**
 * Author: chilli
 * License: CC0
 * Description: z[x] computes the length of the longest common
     prefix of s[i:] and s, except z[0] = 0. (abacaba ->
     0010301)
 * Time: O(n)
 * Status: stress-tested
 */
#pragma once

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;
}
```

DynamicAhoCorasik.h

104 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-18
 * License: CC0
 * Source: marian's (TC) code
 * Description: Deletion happens by creating another aho
     corasik.
 * Aho-Corasick automaton, used for multiple pattern matching.
 * Initialize with AhoCorasick ac(patterns); the automaton
     start node will be at index 0.
 * find(word) returns for each position the index of the
     longest word that ends there, or -1 if none.
 * findAll($-$, word) finds all words (up to $N \sqrt N$ many
     if no duplicate patterns)
 * that start at each position (shortest first).
 * Duplicate patterns are allowed; empty patterns are not.
 * To find the longest words that start at each position,
     reverse all input.
```

```
 * For large alphabets, split each symbol into chunks, with
     sentinel bits for symbol boundaries.
 * Time: construction takes $O(26N)$, where $N =$ sum of length
     of patterns.
 * find(x) is $O(N)$, where N = length of x. findAll is $O(NM)$
    .
 * Status: stress-tested
 */
#pragma once

struct AhoCorasick {
  enum {alpha = 26, first = 'A'}; // change this!
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(next)); }
  };
  vector<Node> N;
  vi backp;
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c : s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

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

vector<string> vc;
vc.push_back(s);
for(int i=0;i<LIM;i++) {
  if(ad[0][i].vs.size()>0) {
    for(auto x: ad[0][i].vs) {
      vc.push_back(x);
    }
    ad[0][i]=Aho();
  }
  else {
    for(auto x: vc) {
      ad[0][i].add(x);
    }
    ad[0][i].build_aho();
    break;
  }
}
```

Number theory (5)

LinearDiophantine.h

79 lines

```
/**
 * Author: Arjo
 * Description: Solving linear diophantine eqns ($ax + by = c$)
     .
 */

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

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

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

int find_all_solutions(int a, int b, int c, int minx, int maxx,
     int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

FastEratosthenes.h

39 lines

```
/**
 * Author: Jakob Kogler, chilli, pajenegod
 * Date: 2020-04-12
 * License: CC0
 * Description: Prime sieve for generating all primes smaller
     than LIM.
 * Time: LIM=1e9 $\approx$ 1.5s
 * Status: Stress-tested
 * Details: Despite its n log log n complexity, segmented sieve
     is still faster
 * than other options, including bitset sieves and linear
     sieves. This is
 * primarily due to its low memory usage, which reduces cache
     misses. This
 * implementation skips even numbers.
 *
 * Benchmark can be found here: https://ideone.com/e7TbX4
 *
 * The line 'for (int i=idx; i<S+L; idx = (i += p))' is done on
     purpose for performance reasons.
 * Se https://github.com/kth-competitive-programming/kactl/pull
     /166#discussion_r408354338
```

```
 */
#pragma once

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 — 26 lines

```
/**
 * Author: chilli, c1729, Simon Lindholm
 * Date: 2019-03-28
 * License: CC0
 * Source: Wikipedia, https://miller-rabin.appspot.com/
 * 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 $a^b \mod c$.
 * Status: Stress-tested
 */
#pragma once

#include "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 — 62 lines

```
/**
 * Author: chilli, SJTU, pajenegod
 * Date: 2020-03-04
 * License: CC0
 * Source: own
 * Description: Pollard-rho randomized factorization algorithm.
     Returns prime
 * factors of a number, in arbitrary order (e.g. 2299 -> \{11,
     19, 11\}).
 * Time: $O(n^{1/4})$, less for numbers with small factors.
 * Status: stress-tested
 *
 * Details: This implementation uses the improvement described
     here
 * (https://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm#
     Variants), where
 * one can accumulate gcd calls by some factor (40 chosen here
     through
 * exhaustive testing). This improves performance by
     approximately 6-10x
 * depending on the inputs and speed of gcd. Benchmark found
     here:
 * (https://ideone.com/nGGD9T)
 *
 * GCD can be improved by a factor of 1.75x using Binary GCD
 * (https://lemire.me/blog/2013/12/26/fastest-way-to-compute-
     the-greatest-common-divisor/).
 * However, with the gcd accumulation the bottleneck moves from
     the gcd calls
 * to the modmul. As GCD only constitutes ~12% of runtime,
     speeding it up
 * doesn't matter so much.
 *
 * This code can probably be sped up by using a faster mod mul
     - potentially
 * montgomery reduction on 128 bit integers.
 * Alternatively, one can use a quadratic sieve for an
     asymptotic improvement,
 * which starts being faster in practice around 1e13.
 *
 * Brent's cycle finding algorithm was tested, but doesn't
     reduce modmul calls
 * significantly.
 *
 * Subtle implementation notes:
 * - we operate on residues in [1, n]; modmul can be proven to
     work for those
 * - prd starts off as 2 to handle the case n = 4; it's
     harmless for other n
 *   since we're guaranteed that n > 2. (Pollard rho has
     problems with prime
 *   powers in general, but all larger ones happen to work.)
 * - t starts off as 30 to make the first gcd check come
     earlier, as an
 *   optimization for small numbers.
 */
#pragma once

#include "ModMulLL.h"
#include "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;
}
```

euclid.h — 15 lines

```
/**
 * Author: Unknown
 * Date: 2002-09-15
 * Source: predates tinyKACTL
 * Description: Finds two integers $x$ and $y$, such that $ax+
     by=\gcd(a,b)$. If
 * you just need gcd, use the built in \texttt{\_\_gcd} instead
     .
 * If $a$ and $b$ are coprime, then $x$ is the inverse of $a \
     pmod{b}$.
 */
#pragma once

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 — 23 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2019-05-22
 * License: CC0
 * Description: Chinese Remainder Theorem.
 *
 * \texttt{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: $\log(n)$
 * Status: Works
 */
#pragma once

#include "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;
}
```

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

Pythagorean triples are uniquely generated by

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

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.
$p = 962592769$ is such that $2^{21} \mid p-1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.
$\sum_{d|n} d = O(n \log \log n)$.

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 $n < 1e19$.

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

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)

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

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

Combinatorics (6)

Cycles

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

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

Derangements

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

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

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

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

Partition function

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

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

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

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

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

Bernoulli numbers

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

Sums of powers:

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

Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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. $\pi(j) > j$.

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

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

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

Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

Bell numbers

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

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

Catalan numbers

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

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

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

Algebra (7)

$$f(x) = f(a) + f'(a)\frac{x-a}{1!} + f''(a)\frac{x-a}{2!} + ...$$

Polynomial.h 24 lines

```
/**
 * Author: David Rydh, Per Austrin
 * Date: 2003-03-16
 * Description:
 */
#pragma once

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

PolyRoots.h (35 lines)

```
/**
 * Author: Per Austrin
 * Date: 2004-02-08
 * License: CC0
 * Description: Finds the real roots to a polynomial.
 * Usage: polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0
 * Time: O(n^2 \log(1/\epsilon))
 */
#pragma once

#include "Polynomial.h"

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

PolyInterpolate.h (25 lines)

```
/**
 * Author: Simon Lindholm
 * Date: 2017-05-10
 * License: CC0
 * Source: Wikipedia
 * 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: O(n^2)
 */
#pragma once

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 (38 lines)

```
/**
 * Author: Lucian Bicsi
 * Date: 2017-10-31
 * License: CC0
 * Source: Wikipedia
 * 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: O(N^2)
 * Status: bruteforce-tested mod 5 for n <= 5 and all s
 */
#pragma once

#include "../number-theory/ModPow.h"

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

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

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

LinearRecurrence.h (44 lines)

```
/**
 * Author: Lucian Bicsi
 * Date: 2018-02-14
 * License: CC0
 * Source: Chinese material
 * Description: Generates the $k$'th term of an $n$-order
 * linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$,
 * given $S[0 \ldots \ge n-1]$ and $tr[0 \ldots n-1]$.
 * Faster than matrix multiplication.
 * Useful together with Berlekamp--Massey.
 * Usage: linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number
 * Time: O(n^2 \log k)
 * Status: bruteforce-tested mod 5 for n <= 5
 */
#pragma once

const ll mod = 5; /** exclude-line */

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

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

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

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

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

SolveLinear.h (49 lines)

```
/**
 * Author: Per Austrin, Simon Lindholm
 * Date: 2004-02-08
 * License: CC0
 * 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: O(n^2 m)
 * Status: tested on kattis:equationsolver, and bruteforce-
      tested mod 3 and 5 for n,m <= 3
 */
#pragma once

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

19 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2016-09-06
 * License: CC0
 * Source: me
 * Description: To get all uniquely determined values of $x$
     back from SolveLinear, make the following changes:
 * Status: tested on kattis:equationsolverplus, stress-tested
 */
#pragma once

#include "SolveLinear.h"

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

SolveLinearBinary.h

46 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2016-08-27
 * License: CC0
 * Source: own work
 * 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: O(n^2 m)
 * Status: bruteforce-tested for n, m <= 4
 */
#pragma once

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)
}
```

FastFourierTransform.h

58 lines

```
/**
 * Author: Ludo Pulles, chilli, Simon Lindholm
 * Date: 2019-01-09
 * License: CC0
 * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf (
     do read, it's excellent)
   Accuracy bound from http://www.daemonology.net/papers/fft.
       pdf
 * Description: fft(a) computes $\hat f(k) = \sum_x a[x] \exp
     (2\pi i \cdot k x / N)$ for all $k$. N must be a power of
      2.
   Useful for convolution:
   \texttt{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 $(\sum a_i^2 + \sum b_i^2)\log_2{N} < 9\
       cdot10^{14}$
   (in practice $10^{16}$; higher for random inputs).
   Otherwise, use NTT/FFTMod.
 * Time: O(N \log N) with $N = |A|+|B|$ ($\tilde 1s$ for $N=2^{
     22}$)
 * Status: somewhat tested
 * Details: An in-depth examination of precision for both FFT
     and FFTMod can be found
 * here (https://github.com/simonlindholm/fft-precision/blob/
     master/fft-precision.md)
 */
#pragma once

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)  /// include-line
      auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];
                    /// exclude-line
      C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
                     /// exclude-line
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

FastFourierTransformMod.h

39 lines

```
/**
 * Author: chilli
 * Date: 2019-04-25
 * License: CC0
 * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf
 * Description: Higher precision FFT, can be used for
     convolutions modulo arbitrary integers
 * as long as $N\log_2N\cdot \text{mod} < 8.6 \cdot 10^{14}$ (
     in practice $10^{16}$ or higher).
 * Inputs must be in $[0, \text{mod})$.
 * Time: O(N \log N), where $N = |A|+|B|$ (twice as slow as NTT
      or FFT)
 * Status: stress-tested
 * Details: An in-depth examination of precision for both FFT
     and FFTMod can be found
 * here (https://github.com/simonlindholm/fft-precision/blob/
     master/fft-precision.md)
 */
#pragma once

#include "FastFourierTransform.h"

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

NumberTheoreticTransform.h

53 lines

```
/**
 * Author: chilli
 * Date: 2019-04-16
 * License: CC0
 * Source: based on KACTL's FFT
```

```
 * 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.
   \texttt{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: O(N \log N)
 * Status: stress-tested
 */
#pragma once

#include "../number-theory/ModPow.h"

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

29 lines

```
/**
 * Author: Lucian Bicsi
 * Date: 2015-06-25
 * License: GNU Free Documentation License 1.2
 * Source: csacademy
 * Description: Transform to a basis with fast convolutions of
     the form
 * $\displaystyle c[z] = \sum\nolimits_{z = x \oplus y} a[x] \
     cdot b[y]$,
 * where $\oplus$ is one of AND, OR, XOR. The size of $a$ must
     be a power of two.
 * Time: O(N \log N)
 * Status: stress-tested
 */
#pragma once

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 ///
            include-line
        // pii(u + v, u - v);                   // XOR ///
            include-line
    }
  }
  // if (inv) for (int& x : a) x /= sz(a); // XOR only ///
      include-line
}
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;
}
```

WalshHadamard.h

20 lines

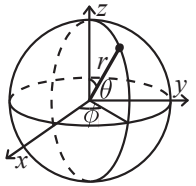
```
/**
 * Author: Arjo
 * Date: Unknown
 * License: CC0
 * Description: $C_k = \sum_{i\otimes{j} = k} A_i B_j$
 * Usage: Apply the transform, point multiply and invert
 * Time: O(N \log N)
 * Status: untested
 */
void WalshHadamard(Poly &P, bool invert) {
  for (int len = 1; 2 * len <= sz(P); len <<= 1) {
    for (int i = 0; i < sz(P); i += 2 * len) {
      rep(j, 0, len) {
        auto u = P[i + j], v = P[i + len + j];
        P[i + j] = u + v, P[i + len + j] = u - v; // XOR
      }
    }
  }
  if (invert) for (auto &x : P) x /= sz(P);
}
```

OnlineFFT.h

28 lines

```
/**
 * Author: Arjo
 * Date: Unknown
 * License: CC0
 * Description: Given $B_1, \ldots B_m$, compute $A_i = \sum_{j
       = 1}^{i - 1} A_j * B_{i - j}$
 * Usage: 1-indexed, pad B[i] = 0 for i > m
 * Time: O(N \log^2 N)
 * Status: untested
 */
#pragma once
void online(const Poly &B, CD a1, int n, Poly &A) {
  const int m = SZ(B) - 1;
  A.assign(n + 1, 0); A[1] = a1;
  auto bst = B.begin(), ast = A.begin();
  REP(i, 1, n) {
    A[i + 1] += A[i] * B[1];
    if (i + 2 <= n) A[i + 2] += A[i] * B[2];
    for (int pw = 2; i % pw == 0 && pw + 1 <= m; pw <<= 1) {
      Poly blockA(ast + i - pw, ast + i);
      Poly blockB(bst + pw + 1, bst + min(pw * 2, m) + 1);
      Poly prod = conv(blockA, blockB);
      REP(j, 0, sz(prod)) {
        if (i + 1 + j <= n)
          A[i + 1 + j] += prod[j];
      }
    }
  }
}
```

Geometry (8)



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

Point.h

56 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-02-26
 * License: CC0
 * Source: My head with inspiration from tinyKACTL
 * Description: Class to handle points in the plane.
 *  T can be e.g. double or long long. (Avoid int.)
 * Status: Works fine, used a lot
 */
#pragma once
#include <bits/stdc++.h>

template <class T> int sgn(T x) { return (x > T(0)) - (x < T(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; }
  // abs() == dist()
  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(); }
  P translate(P v) { return P(x + v.x, y + v.y); }
```

```
  // scale an object by a certain ratio alpha around a
  // center c, we need to shorten or lengthen the vector
  // from c to every point by a factor alpha, while
  // conserving the direction
  P scale(P c, double factor) { return c + (*this - c) * factor
      ; }
  // 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 << ")";
  }

  // Additional random shit
  bool isPerp(P p) { return P(x, y).dot(p) == 0; }
  double angle(P p) {
    double costheta = P(x, y).dot(p) / (*this).dist() / p.dist
        ();
    return acos(fmax(-1.0, fmin(1.0, costheta)));
  }
  T orient(P b, P c) { return (b - *this).cross(c - *this); }
};
```

lineDistance.h — 22 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source: Basic math
 * Description:\\
\begin{minipage}{75mm}
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.
\end{minipage}
\begin{minipage}{15mm}
\includegraphics[width=\textwidth]{content/geometry/
     lineDistance}
\end{minipage}
 * Status: tested
 */
#pragma once

#include "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 — 28 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source:
 * Description:\\
\begin{minipage}{75mm}
Returns the shortest distance between point p and the line
     segment from point s to e.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-10mm}
\includegraphics[width=\textwidth]{content/geometry/
     SegmentDistance}
\end{minipage}
 * Usage:
 *  Point<double> a, b(2,2), p(1,1);
 *  bool onSegment = segDist(a,b,p) < 1e-10;
 * Status: tested
 */
#pragma once

#include "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 — 39 lines

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-27
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description:\\
\begin{minipage}{75mm}
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.
\end{minipage}
\begin{minipage}{15mm}
\includegraphics[width=\textwidth]{content/geometry/
     SegmentIntersection}
\end{minipage}
 * Usage:
 * vector<P> inter = segInter(s1,e1,s2,e2);
 * if (sz(inter)==1)
 *   cout << "segments intersect at " << inter[0] << endl;
 * Status: stress-tested, tested on kattis:intersection
 */
#pragma once

#include "Point.h"
#include "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 {all(s)};
}
```

lineIntersection.h — 33 lines

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-05-05
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description:\\
\begin{minipage}{75mm}
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.
\end{minipage}
\begin{minipage}{15mm}
\includegraphics[width=\textwidth]{content/geometry/
     lineIntersection}
\end{minipage}
 * Usage:
 *  auto res = lineInter(s1,e1,s2,e2);
 *  if (res.first == 1)
 *    cout << "intersection point at " << res.second << endl;
 * Status: stress-tested, and tested through half-plane tests
 */
#pragma once

#include "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 — 23 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source:
 * 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;
 * Status: tested
 */
#pragma once

#include "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

16 lines

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Returns true iff p lies on the line segment
     from s to e.
 * Use \texttt{(segDist(s,e,p)<=epsilon)} instead when using
     Point<double>.
 * Status:
 */
#pragma once

#include "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;
}
```

linearTransformation.h

26 lines

```
/**
 * Author: Per Austrin, Ulf Lundstrom
 * Date: 2009-04-09
 * License: CC0
 * Source:
 * Description:\\
\begin{minipage}{75mm}
 Apply the linear transformation (translation, rotation and
     scaling) which takes line p0-p1 to line q0-q1 to point r.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-8mm}
\includegraphics[width=\textwidth]{content/geometry/
    linearTransformation}
\vspace{-2mm}
\end{minipage}
 * Status: not tested
 */
#pragma once

#include "Point.h"

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

LineProjectionReflection.h

21 lines

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-10-29
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Projects point p onto line ab. Set refl=true to
     get reflection
 * of point p across line ab insted. The wrong point will be
     returned if P is
 * an integer point and the desired point doesn't have integer
     coordinates.
 * Products of three coordinates are used in intermediate steps
     so watch out
 * for overflow.
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

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

Angle.h

51 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2015-01-31
 * License: CC0
 * Source: me
 * Description: A class for ordering angles (as represented by
     int points and
 *  a number of rotations around the origin). Useful for
     rotational sweeping.
 *  Sometimes also represents points or vectors.
 * Usage:
 *  vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
 *  int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
 *  // sweeps j such that (j-i) represents the number of
     positively oriented triangles with vertices at 0 and i
 * Status: Used, works well
 */
#pragma once

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

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

CircleIntersection.h

22 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2015-09-01
 * License: CC0
 * Description: Computes the pair of points at which two
     circles intersect. Returns false in case of no
     intersection.
 * Status: stress-tested
 */
#pragma once

#include "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;
}
```

CircleTangents.h

29 lines

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-10-31
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Finds the external tangents of two circles, or
     internal if r2 is negated.
 * Can return 0, 1, or 2 tangents -- 0 if one circle contains
     the other (or overlaps it, in the internal case, or if
     the circles are the same);
 * 1 if the circles are tangent to each other (in which case .
     first = .second and the tangent line is perpendicular to
     the line between the centers).
 * .first and .second give the tangency points at circle 1 and
     2 respectively.
 * To find the tangents of a circle with a point set r2 to 0.
 * Status: tested
 */
#pragma once

#include "Point.h"

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

```
  return out;
}
```

CircleLine.h

24 lines

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-10-29
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Finds the intersection between a circle and a
     line.
 * Returns a vector of either 0, 1, or 2 intersection points.
 * P is intended to be Point<double>.
 * Status: unit tested
 */

#pragma once

#include "Point.h"

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

CirclePolygonIntersection.h

33 lines

```
/**
 * Author: chilli, Takanori MAEHARA
 * Date: 2019-10-31
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/
     master/geometry/_geom.cc#L744
 * Description: Returns the area of the intersection of a
     circle with a
 * ccw polygon.
 * Time: O(n)
 * Status: Tested on GNYR 2019 Gerrymandering, stress-tested
 */
#pragma once

#include "../../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;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

circumcircle.h

28 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-04-11
 * License: CC0
 * Source: http://en.wikipedia.org/wiki/Circumcircle
 * Description:\\
\begin{minipage}{75mm}
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.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-2mm}
\includegraphics[width=\textwidth]{content/geometry/
     circumcircle}
\end{minipage}
 * Status: tested
 */
#pragma once

#include "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;
}
```

MinimumEnclosingCircle.h

30 lines

```
/**
 * Author: Andrew He, chilli
 * Date: 2019-05-07
 * License: CC0
 * Source: folklore
 * Description: Computes the minimum circle that encloses a set
      of points.
 * Time: expected O(n)
 * Status: stress-tested
 */
#pragma once

#include "circumcircle.h"

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};
}
```

InsidePolygon.h

31 lines

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * 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.
 * Time: O(n)
 * Usage:
 * vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
 * bool in = inPolygon(v, P{3, 3}, false);
 * Status: stress-tested and tested on kattis:pointinpolygon
 */
#pragma once

#include "Point.h"
#include "OnSegment.h"
#include "SegmentDistance.h"

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

19 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source: tinyKACTL
 * Description: Returns twice the signed area of a polygon.
 *  Clockwise enumeration gives negative area. Watch out for
      overflow if using int as T!
 * Status: Stress-tested and tested on kattis:polygonarea
 */
#pragma once

#include "Point.h"

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

22 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-04-08
 * License: CC0
 * Source:
 * Description: Returns the center of mass for a polygon.
 * Time: O(n)
 * Status: Tested
 */
```

```cpp
#pragma once

#include "Point.h"

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

PolygonCut.h

37 lines

```cpp
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source:
 * Description:\\
\begin{minipage}{75mm}
 Returns a vector with the vertices of a polygon with
      everything to the left of the line going from s to e cut
      away.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-6mm}
\includegraphics[width=\textwidth]{content/geometry/PolygonCut}
\vspace{-6mm}
\end{minipage}
 * Usage:
 *  vector<P> p = ...;
 *  p = polygonCut(p, P(0,0), P(1,0));
 * Status: tested but not extensively
 */
#pragma once

#include "Point.h"
#include "lineIntersection.h"

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

PolygonUnion.h

49 lines

```cpp
/**
 * Author: black_horse2014, chilli
 * Date: 2019-10-29
 * License: Unknown
 * Source: https://codeforces.com/gym/101673/submission
      /50481926
 * Description: Calculates the area of the union of $n$
      polygons (not necessarily
 * convex). The points within each polygon must be given in CCW
       order.
 * (Epsilon checks may optionally be added to sideOf/sgn, but
      shouldn't be needed.)
 * Time: $O(N^2)$, where $N$ is the total number of points
 * Status: stress-tested, Submitted on ECNA 2017 Problem A
 */
#pragma once

#include "Point.h"
#include "sideOf.h"

typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

ConvexHull.h

35 lines

```cpp
/**
 * Author: Stjepan Glavina, chilli
 * Date: 2019-05-05
 * License: Unlicense
 * Source: https://github.com/stjepang/snippets/blob/master/
      convex_hull.cpp
 * Description:
\\\begin{minipage}{75mm}
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.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-6mm}
\includegraphics[width=\textwidth]{content/geometry/ConvexHull}
\vspace{-6mm}
\end{minipage}
 * Time: O(n \log n)
 * Status: stress-tested, tested with kattis:convexhull
*/
#pragma once

#include "Point.h"

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

25 lines

```cpp
/**
 * Author: Oleksandr Bacherikov, chilli
 * Date: 2019-05-05
 * License: Boost Software License
 * Source: https://codeforces.com/blog/entry/48868
 * Description: Returns the two points with max distance on a
      convex hull (ccw,
 * no duplicate/collinear points).
 * Status: stress-tested, tested on kattis:roberthood
 * Time: O(n)
 */
#pragma once
#include "Point.h"

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

32 lines

```cpp
/**
 * Author: chilli
 * Date: 2019-05-17
 * License: CC0
 * Source: https://github.com/ngthanhtrung23/ACM_Notebook_new
 * 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.
 * Usage:
 * Status: stress-tested
 * Time: O(\log N)
 */
#pragma once

#include "Point.h"
#include "sideOf.h"
#include "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 (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;
}
```

LineHullIntersection.h

62 lines

```
/**
 * Author: Oleksandr Bacherikov, chilli
 * Date: 2019-05-07
 * License: Boost Software License
 * Source: https://github.com/AlCash07/ACTL/blob/master/include
     /actl/geometry/algorithm/intersect/line_convex_polygon.
     hpp
 * 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:
 *  \begin{itemize*}
 *    \item $(-1, -1)$ if no collision,
 *    \item $(i, -1)$ if touching the corner $i$,
 *    \item $(i, i)$ if along side $(i, i+1)$,
 *    \item $(i, j)$ if crossing sides $(i, i+1)$ and $(j, j+1)
     $.
 *  \end{itemize*}
 *  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.
 * \texttt{extrVertex} returns the point of a hull with the max
      projection onto a line.
 * Time: O(\log n)
 * Status: stress-tested
 */
#pragma once

#include "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 = sz(poly), 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;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    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] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

ClosestPair.h

30 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2019-04-17
 * License: CC0
 * Source: https://codeforces.com/blog/entry/58747
 * Description: Finds the closest pair of points.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

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;
}
```

ManhattanMST.h

38 lines

```
/**
 * Author: chilli, Takanori MAEHARA
 * Date: 2019-11-02
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/
     master/geometry/rectilinear_mst.cc
 * Description: Given N points, returns up to 4*N edges, which
     are guaranteed
 * to contain a minimum spanning tree for the graph with edge
     weights w(p, q) =
 * |p.x - q.x| + |p.y - q.y|. Edges are in the form (distance,
     src, dst). Use a
 * standard MST algorithm on the result to find the final MST.
 * Time: O(N \log N)
 * Status: Stress-tested
 */
#pragma once
#include "Point.h"

typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
  vi id(sz(ps));
  iota(all(id), 0);
  vector<array<int, 3>> edges;
  rep(k,0,4) {
    sort(all(id), [&](int i, int j) {
         return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
    map<int, int> sweep;
    for (int i : id) {
      for (auto it = sweep.lower_bound(-ps[i].y);
                it != sweep.end(); sweep.erase(it++)) {
        int j = it->second;
        P d = ps[i] - ps[j];
        if (d.y > d.x) break;
        edges.push_back({d.y + d.x, i, j});
      }
      sweep[-ps[i].y] = i;
    }
    for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
  }
  return edges;
}
```

kdTree.h

74 lines

```
/**
 * Author: Stanford
 * Date: Unknown
 * Source: Stanford Notebook
 * Description: KD-tree (2d, can be extended to 3d)
 * Status: Tested on excellentengineers
 */
#pragma once

#include "Point.h"

typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
```

```
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

DelaunayTriangulation.h

25 lines

```
/**
 * Author: Mattias de Zalenski
 * Date: Unknown
 * Source: Geometry in C
 * Description: Computes the Delaunay triangulation of a set of
      points.
 *  Each circumcircle contains none of the input points.
 *  If any three points are collinear or any four are on the
     same circle, behavior is undefined.
 * Time: O(n^2)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"
#include "3dHull.h"

template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

FastDelaunay.h

106 lines

```
/**
 * Author: Philippe Legault
 * Date: 2016
 * License: MIT
 * Source: https://github.com/Bathlamos/delaunay-triangulation/
 * Description: Fast Delaunay triangulation.
 * Each circumcircle contains none of the input points.
 * There must be no duplicate points.
 * If all points are on a line, no triangles will be returned.
 * Should work for doubles as well, though there may be
     precision issues in 'circ'.
 * Returns triangles in order \{t[0][0], t[0][1], t[0][2], t
     [1][0], \dots\}, all counter-clockwise.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

PolyhedronVolume.h

14 lines

```
/**
 * Author: Mattias de Zalenski
 * Date: 2002-11-04
 * Description: Magic formula for the volume of a polyhedron.
     Faces should point outwards.
 * Status: tested
 */
#pragma once

template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
  double v = 0;
  for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
}
```

Point3D.h

44 lines

```
/**
 * Author: Ulf Lundstrom with inspiration from tinyKACTL
 * Date: 2009-04-14
 * License: CC0
 * Source:
 * Description: Class to handle points in 3D space.
 *  T can be e.g. double or long long.
 * Usage:
 * Status: tested, except for phi and theta
 */
#pragma once
```

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

3dHull.h — 63 lines

```
/**
 * Author: Johan Sannemo
 * Date: 2017-04-18
 * Source: derived from https://gist.github.com/msg555/4963794
     by Mark Gordon
 * Description: Computes all faces of the 3-dimension hull of a
      point set.
 *  *No four points must be coplanar*, or else random results
     will be returned.
 *  All faces will point outwards.
 * Time: O(n^2)
 * Status: tested on SPOJ CH3D
 */
#pragma once

#include "Point3D.h"

typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

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

sphericalDistance.h — 18 lines

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-04-07
 * License: CC0
 * Source: My geometric reasoning
 * Description: Returns the shortest distance on the sphere
     with radius radius between the points with azimuthal
     angles (longitude) f1 ($\phi_1$) and f2 ($\phi_2$) from x
      axis and zenith angles (latitude) t1 ($\theta_1$) and t2
      ($\theta_2$) from z axis (0 = north pole). All angles
     measured in radians. The algorithm starts by converting
     the spherical coordinates to cartesian coordinates so if
     that is what you have you can use only the two last rows.
      dx*radius is then the difference between the two points
     in the x direction and d*radius is the total distance
     between the points.
 * Status: tested on kattis:airlinehub
 */
#pragma once

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

Various (9)

TernarySearch.h — 27 lines

```
/**
 * Author: Simon Lindholm
 * Date: 2015-05-12
 * License: CC0
 * Source: own work
 * Description:
 * Find the smallest i in $[a,b]$ that maximizes $f(i)$,
     assuming that $f(a) < \dots < f(i) \ge \dots \ge f(b)$.
 * To reverse which of the sides allows non-strict inequalities
     , change the < marked with (A) to <=, and reverse the
     loop at (B).
 * To minimize $f$, change it to >, also at (B).
 * Usage:
  int ind = ternSearch(0,n-1,[\&](int i){return a[i];});
 * Time: O(\log(b-a))
 * Status: tested
 */
#pragma once

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

Convolution.h — 48 lines

```
/**
 * Author: Amul Agrawal
 * Description: Getting different convolutions
 * Time: O(n 2^n)
 */
#pragma once

// Zeta/SOS, N*2^N
rep(i,0,M)
  for(int mask = (1<<M) - 1; mask >= 0; mask--)
    if((mask>>i)&1)
      F[mask] += F[mask ^ (1 << i)];
// Rev mask loop and invert bit condition for superset sum

// Base from SOS
for(int i = M - 1; i >= 0; i--)
    for(int mask = (1 << M) - 1; mask >= 0; mask--)
      if((mask >> i)&1)
        F[mask] -= F[mask ^ (1 << i)];
// Rev mask loop and invert condition for base from Sum over
     superset

// Mobius, F[s] = SUM(-1^{s/s'} * F[s']), N*2^N
// F[1011] = F[1011] - F[0011] - F[1001] - F[1010] + F[1000]
     ...
rep(i,0,M) rep(mask, 0, 1<<M) if((mask>>i)&1)
      F[mask] -= F[mask ^ (1 << i)];

// sos(mu(f(x))) = f(x) = mu(sos(f(x)))

// fog[s] = SUM(f[s']*g[s/s']), N^2 * 2^N
// Make fhat[][] = {0} and ghat[][] = {0}
rep(mask,0,1<<N) {
    fhat[__builtin_popcount(mask)][mask] = f[mask];
```

```
    ghat[__builtin_popcount(mask)][mask] = g[mask];
}
// Apply zeta transform on fhat[][] and ghat[][]
rep(i,0,N+1) rep(j,0,N) rep(mask,0,1<<N) if((mask>>j)&1) {
  fhat[i][mask] += fhat[i][mask ^ (1 << j)];
  ghat[i][mask] += ghat[i][mask ^ (1 << j)];
}
// Do the convolution and store into h[][] = {0}
rep(mask,0,(1<<N)) rep(i,0,N+1) rep(j,0,i+1)
          h[i][mask] += fhat[j][mask] * ghat[i - j][mask];
// Apply inverse SOS dp on h[][]
rep(i,0,N+1) rep(j,0,N) rep(mask,0,1<<N) if((mask>>j)&1)
  h[i][mask] -= h[i][mask ^ (1 << j)];

rep(mask,0,1<<N) fog[mask] = h[__builtin_popcount(mask)][mask];
```

PolyModPoly.h

39 lines

```
/**
 * Author: BenQ
 * Date: 2022-10-3
 * Description: Poly Mod Poly
 */
#pragma once
#define rsz resize
poly RSZ(poly p, int x) { p.rsz(x); return p; }
poly rev(poly p) { reverse(all(p)); return p; }
poly inv(poly A, int n) { // Q-(1/Q-A)/(-Q^{-2})
  poly B{1/A[0]};
  while (sz(B) < n) {
    int x = 2*sz(B);
    B = RSZ(2*B-conv(RSZ(A,x),conv(B,B)),x); } // fft
  return RSZ(B,n);
}
pair<poly,poly> divi(const poly& f, const poly& g) {
  if (sz(f) < sz(g)) return {{},f};
  auto q = mul(inv(rev(g),sz(f)-sz(g)+1),rev(f));
  q = rev(RSZ(q,sz(f)-sz(g)+1));
  auto r = RSZ(f-mul(q,g),sz(g)-1); return {q,r};
}
typedef vector<mi> vmi; // mi = modular int
struct MultipointEval {
  poly stor[1<<18];
  void prep(vmi v, int ind = 1) { // v -> places to evaluate at
    if (sz(v) == 1) { stor[ind] = {-v[0],1}; return; }
    int m = sz(v)/2;
    prep(vmi(begin(v),begin(v)+m),2*ind);
    prep(vmi(m+all(v)),2*ind+1);
    stor[ind] = conv(stor[2*ind],stor[2*ind+1]);
  }
  vmi res;
  void eval(vmi v, int ind = 1) {
    v = divi(v,stor[ind]).s;
    if (sz(stor[ind]) == 2) { res.pb(v[0]); return; }
    eval(v,2*ind); eval(v,2*ind+1);
  }
};
```

PollardRho.h

21 lines

```
// Pollard Rho: O(N^(1/4))
ull pollard(ull n){
  auto f = [n](ull x){return (mod_mull(x,x,n)+1)%n;};
  if(!(n&1)) return 2;
  for(ull i=2;;i++){
    ull x=i;y=f(x),p;
    while((p=__gcd(n+y-x,n)) == 1)
      x=f(x), y=f(f(y));
    if(p!=n) return p;
  }
}
vector <ull> factor(ull n) {
  if (n==1) return {};
  if (isPrime(n)){ //use miller rabin
    return{n};
  }
  ull x=pollard(n);
  auto l = factor(x), r = factor(n/x);
  l.insert(l.end(),all(r));
  return l;
}
```

makefile.txt

53 lines

```
Make File:
cc = g++
flags = -g \
  -DFLUX \
  -pedantic \
  -Wall \
  -Wextra \
  -Wshadow \
  -Wfloat-equal \
  -Wconversion \
  -Wlogical-op \
  -Wshift-overflow=2 \
  -Wduplicated-cond \
  -Wcast-qual \
  -Wcase-align \
  -D_GLIBCXX_DEBUG \
  -D_GLIBCXX_DEBUG_PEDANTIC \
  -fsanitize=undefined \
  -fno-sanitize-recover \
  -fstack-protector \
  -std=c++17 \

libs =

bin/%: bin/%.o
  $(cc) -o $@ $< $(flags)

bin/%.o: src/%.cpp
  $(cc) -o $@ -c $< $(flags)

clean:
  rm bin/*

Optimization Tricks:
Bit Hacks: x & -x is least bit in x
for (int x=m;x;){--x&=m;....}
loops over all subset masks of m except m.
c = x&-x, r=x+c;((r^x)>>2)/c
next no after x with same number of bits sets
rep(b,0,k) rep(i,0,(1<<k)
  if (i & 1<<b) D[i] += D[i^(1<<b)]
computes all sums of subsets.

Pragmas:

#pragma GCC optimize("Ofast")
#pragma GCC optimize("O1")
#pragma GCC optimize("O2")
#pragma GCC optimize("O3")
#pragma GCC optimize("Os")

#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
```

PersistentSegtree.h

46 lines

```
// PersistentSegmentTree.h
// Description: Zero-indexed max-tree. Bounds are inclusive to
     the left and exclusive to the right. Can be changed by
     modifying T, f and unit.
// Time: O(log N)

struct node
{
    int count;
    node *left, *right;
    node(int count, node *left, node *right) : count(count),
         left(left), right(right) {}
    node* insert(int l, int r, int w);
};

node *null = new node(0, NULL, NULL); // see line 135
node* node::insert(int l, int r, int w)
{
    if(l <= w && w < r)
    {
        if(l+1 == r)
            return new node(this->count+1, null, null);
        int m = (l+r)>>1;
        return new node(this->count+1, this->left->insert(l, m,
             w), this->right->insert(m, r, w));
    }
    return this;
}

int query(node *a, node *b, int l, int r, int k)
{
    if(l+1 == r)
        return l;
    int m = (l+r)>>1;
    int count = a->left->count - b->left->count;
    if(count >= k)
        return query(a->left, b->left, l, m, k);
    return query(a->right, b->right, m, r, k - count);
}

int a[N], RM[N];
node *root[N];

// while generating
null->left = null->right = null;

for(int i = 0; i < n; ++i)
{
    root[i] = (i == 0 ? null : root[i-1])->insert(0, maxi, M[a[
         i]]);
}
```