Ivan Franko National University of Lviv

Stallions

Maksym Shcherba, Petro Tarnavskyi, Yarema Stiahar

47th Annual World Championship of the International Collegiate Programming Contest

March 24, 2024

Contest (1)

template.hpp
26 lines

// hash = 85ed39

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

#define FOR(i, a, b) for(int i = (a); i < (b); i++)
#define RFOR(i, a, b) for(int i = (a) - 1; i >= (b); i--)
#define SZ(a) int(a.size())
#define ALL(a) a.begin(), a.end()
#define PB push_back
#define MP make_pair
#define F first
#define S second

typedef long long LL;
typedef vector<int> VI;
typedef pair<int, int> PII;
typedef double db;

int main()
{
  ios::sync_with_stdio(0);
  cin.tie(0);

  return 0;
}
```

compilation.txt
2 lines

```
g++ -O2 -std=c++20 -Wno-unused-result -Wshadow -Wall -o %e %e.cpp
g++ -std=c++20 -Wshadow -Wall -o %e %e.cpp -fsanitize=address -fsanitize=
     undefined -D_GLIBCXX_DEBUG -g
```

s.sh
6 lines

```
for((i = 0; ; i++)) do
  echo $i
  ./gen $i > in
  diff -w <(./a < in) <(./brute < in) || break
  [ $? == 0 ] || break
done
```

hash.sh
1 lines

```
cpp -dD -P -fpreprocessed $1 | tr -d '[:space:]'| md5sum |cut -c-6
```

Rules

Reject incorrect solutions from your teammates. Try to find counterexamples.

Discuss implementation and try to simplify the solution.

Avoid getting stuck on the problem.

Regularly discuss how many problems need to be solved and what steps to take, starting from the middle of the contest.

At the end of the contest, try to find a problem with an easy implementation.

Troubleshoot

Pre-submit

F9. Create a few manual test cases. Calculate time and memory complexity. Check the limits. Be careful with overflows, constants, clearing mutitestcases, uninitialized variables.

Wrong answer

F9. Print your solution! Read your code. Check pre-submit. Are you sure your algorithm works? Think about precision errors and hash collisions. Have you understood the problem correctly? Write the brute and the generator.

Runtime error

F9. Print your solution! Read your code. F9 with generator. Memory limit exceeded.

Time limit exceeded

What is the complexity of your algorithm? Are you copying a lot of unnecessary data? (References) Do you have any infinite loops? Use arrays, unordered maps instead of vectors and maps.

Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.

- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.

```
                                     \       ,
                                     |\.--._/|
                                    /\ )  )\\/
                                   /(   \  / \
                                  /(   J `(   \
                                 / ) | _\     /
                                /|)  \  eJ    L
                               |  \ L \   L   L
                              /  \  J  `. J   L
                              |  )   L   \/   \
                             /  \    J   (\   /
           _.....___        |  \      \   \```
    ,.._.-'        '''--...-||\     -. \   \
  .'.=.'          4    4  77777       `.\ [ Y
 /   /            4    4      7         \]  J
Y / Y             444444     7           Y   L
| | |          \      4     7            |   L
| | |           Y      4    7            A  J
|   I           |      4    7           /I\ /
|    \          I           \          ( |]/|
J     \         /._         /          -tI/ |
 L     )       /   /'-------'J           `'-:.
 J   .'      ,'  ,' ,     \   `'-.__          \
  \ T      ,'  ,'   )\    /|        ';'---7   /
   \|    ,'L  Y...-' / _.' /        \   /   /
    J   Y  |  J    .'-'   /        ,--.(   /
     L  |  J   L -'     .'        /  |    /\
     |  J.  L  J     .-;.-/      |    \ .' /
     J   L`-J   L____,.-'`       |  _.-'   |
      L  J   L  J                 ``  J    |
      J   L  |   L                    J    |
       L  J  L    \                   L    \
       |   L  ) _.'\                   ) _.'\
       L    \('`    \                 ('`    \
        ) _.'\`-....'                  `-....'
       ('`    \
        `-.___/
```

Data Structures (2)

dsu.hpp

25926a, 31 lines

```
struct DSU
{
  int n;
  VI p, sz;

  void init(int _n)
  {
    n = _n;
    p.resize(n);
    iota(ALL(p), 0);
    sz.assign(n, 1);
  }
  int find(int v)
  {
    if (v == p[v])
      return v;
    return p[v] = find(p[v]);
  }
  bool unite(int u, int v)
  {
    u = find(u);
    v = find(v);
    if (u == v)
      return false;
    if (sz[u] > sz[v])
      swap(u, v);
    p[u] = v;
    sz[v] += sz[u];
    return true;
  }
};
```

fenwick.hpp

319477, 45 lines

```
struct Fenwick
{
  int n;
  vector<LL> t;

  void init(int _n)
  {
    n = _n;
    t.clear();
    t.assign(n, 0);
  }

  void upd(int i, int x)
  {
    for (; i < n; i |= i + 1)
      t[i] += x;
  }

  LL query(int i)
  {
    LL ans = 0;
    for (; i >= 0; i = (i & (i + 1)) - 1)
      ans += t[i];
    return ans;
  }

  // returns n if sum(a) < x
  int lowerBound(LL x)
  {
    LL sum = 0;
    int i = -1;
    int lg = 31 - __builtin_clz(n);
    while (lg >= 0)
    {
      int j = i + (1 << lg);
      if (j < n && sum + t[j] < x)
      {
        sum += t[j];
        i = j;
      }
      lg--;
    }
    return i + 1;
  }
};
```

Minimum on a Segment

Maintain two Fenwick trees with $n = 2^k$ — one for the original array and the other for the reversed array. If $n > 1$, you can use: $n = 1 \ll (32 - \_\_builtin\_clz(n-1))$.

When querying for the minimum on the segment, only consider segments $[(i\&(i+1)), i]$ that are completely inside $[l, r]$.

Add on a Segment

Maintain two Fenwick trees: tMult and tAdd.

To add x on the segment $[l, r]$, tMult.upd(l, x), tMult.upd(r, $-x$), tAdd.upd(l, $-x \cdot (l-1)$), tAdd.upd(r, $x \cdot r$).

$r\cdot$ tMult.query(r) + tAdd.query(r) is the sum on $[0, r]$.

treap.hpp

Description: uncomment in split for explicit key or in merge for implicit priority. Minimum and reverse queries.

7b6f83, 148 lines

```
mt19937 rng;

struct Node
{
  int l, r;
  int x, y;
  int cnt, par;
  int rev, mn;

  Node(int value)
  {
    l = r = -1;
    x = value;
    y = rng();
    cnt = 1;
    par = -1;
    rev = 0;
    mn = value;
  }
};

struct Treap
{
  vector<Node> t;
  void init(int n)
  {
    t.clear();
    t.reserve(n);
  }

  int getCnt(int v)
  {
    if (v == -1)
      return 0;
    return t[v].cnt;
  }
  int getMn(int v)
  {
    if (v == -1)
      return INF;
    return t[v].mn;
  }
  int newNode(int val)
  {
    t.PB({val});
    return SZ(t) - 1;
  }
  void upd(int v)
  {
    if (v == -1)
      return;
    // important!
    t[v].cnt = getCnt(t[v].l) +
    getCnt(t[v].r) + 1;

    t[v].mn = min(t[v].x, min(getMn(t[v].l), getMn(t[v].r)));
  }
  void reverse(int v)
  {
    if (v == -1)
      return;
    t[v].rev ^= 1;
  }
  void push(int v)
  {
    if (v == -1 || t[v].rev == 0)
      return;
    reverse(t[v].l);
    reverse(t[v].r);
    swap(t[v].l, t[v].r);
    t[v].rev = 0;
  }
  PII split(int v, int cnt)
  {
    if (v == -1)
      return {-1, -1};
    push(v);
    int left = getCnt(t[v].l);
    PII res;
    // elements a[v].x == val will be in right part
    // if (val <= a[v].x)
```

```
    if (cnt <= left)
    {
      if (t[v].l != -1)
        t[t[v].l].par = -1;
      // res = split(a[v].l, val);
      res = split(t[v].l, cnt);
      t[v].l = res.S;
      if (res.S != -1)
        t[res.S].par = v;
      res.S = v;
    }
    else
    {
      if (t[v].r != -1)
        t[t[v].r].par = -1;
      // res = split(a[v].r, val);
      res = split(t[v].r, cnt - left - 1);
      t[v].r = res.F;
      if (res.F != -1)
        t[res.F].par = v;
      res.F = v;
    }
    upd(v);
    return res;
  }
  int merge(int v, int u)
  {
    if (v == -1) return u;
    if (u == -1) return v;
    int res;
    // if ((int)(rng() % (getCnt(v) + getCnt(u))) < getCnt(v))
    if (t[v].y > t[u].y)
    {
      push(v);
      if (t[v].r != -1)
        t[t[v].r].par = -1;
      res = merge(t[v].r, u);
      t[v].r = res;
      if (res != -1)
        t[res].par = v;
      res = v;
    }
    else
    {
      if(t[u].l != -1)
        t[t[u].l].par = -1;
      res = merge(v, t[u].l);
      t[u].l = res;
      if (res != -1)
        t[res].par = u;
      res = u;
    }
    upd(res);
    return res;
  }
  // returns index of element [0, n)
  int getIdx(int v, int from = -1)
  {
    if (v == -1)
      return 0;
    int x = getIdx(t[v].par, v);
    push(v);
    if (from == -1 || t[v].r == from)
      x += getCnt(t[v].l) + (from != -1);
    return x;
  }
};
```

lct.hpp

Description: Link-Cut Tree. Given a function $f(1 \dots N) \to 1 \dots N$, evaluates $f^b(a)$ for any a, b. `sz` is for path queries; `sub`, `vsub` are for subtree queries. `x->access()` brings `x` to the top and propagates it; its left subtree will be the path from `x` to the root and its right subtree will be empty. Then `sub` will be the number of nodes in the connected component of `x` and `vsub` will be the number of nodes under `x`. Use `makeRoot` for arbitrary path queries.
Usage: `FOR(i,1,N+1)LCT[i]=new snode(i); link(LCT[1],LCT[2],1);`
Time: $\mathcal{O}(\log N)$

1b67e4, 95 lines

```
typedef struct snode* sn;
struct snode {
  sn p, c[2]; // parent, children
  bool flip = 0; // subtree flipped or not
  int val, sz; // value in node, # nodes in current splay tree
  int sub, vsub = 0; // vsub stores sum of virtual children

  snode(int _val) : val(_val) {
    p = c[0] = c[1] = NULL; calc(); }

  friend int getSz(sn x) { return x?x->sz:0; }
  friend int getSub(sn x) { return x?x->sub:0; }

  void prop() { // lazy prop
    if (!flip) return;
    swap(c[0],c[1]); flip = 0;
    FOR(i,0,2) if (c[i]) c[i]->flip ^= 1;
  }
  void calc() { // recalc vals
    FOR(i,0,2) if (c[i]) c[i]->prop();
    sz = 1+getSz(c[0])+getSz(c[1]);
    sub = 1+getSub(c[0])+getSub(c[1])+vsub;
  }
  int dir() {
    if (!p) return -2;
    FOR(i,0,2) if (p->c[i] == this) return i;
    return -1; // p is path-parent pointer
  } // -> not in current splay tree
  // test if root of current splay tree
  bool isRoot() { return dir() < 0; }
  friend void setLink(sn x, sn y, int d) {
    if (y) y->p = x;
    if (d >= 0) x->c[d] = y; }
  void rot() { // assume p and p->p propagated
    assert(!isRoot()); int x = dir(); sn pa = p;
    setLink(pa->p, this, pa->dir());
    setLink(pa, c[x^1], x); setLink(this, pa, x^1);
    pa->calc();
  }
  void splay() {
    while (!isRoot() && !p->isRoot()) {
      p->p->prop(), p->prop(), prop();
      dir() == p->dir() ? p->rot() : rot();
      rot();
    }
    if (!isRoot()) p->prop(), prop(), rot();
    prop(); calc();
  }
  sn fbo(int b) { // find by order
    prop(); int z = getSz(c[0]); // of splay tree
    if (b == z) { splay(); return this; }
    return b < z ? c[0]->fbo(b) : c[1] -> fbo(b-z-1);
  }
  void access() { // bring this to top of tree, propagate
    for (sn v = this, pre = NULL; v; v = v->p) {
      v->splay(); // now switch virtual children
      if (pre) v->vsub -= pre->sub;
      if (v->c[1]) v->vsub += v->c[1]->sub;
      v->c[1] = pre; v->calc(); pre = v;
    }
    splay(); assert(!c[1]); // right subtree is empty
  }
  void makeRoot() {
    access(); flip ^= 1; access(); assert(!c[0] && !c[1]); }
  friend sn lca(sn x, sn y) {
    if (x == y) return x;
    x->access(), y->access(); if (!x->p) return NULL;
    x->splay(); return x->p?:x; // y was below x in latter case
  } // access at y did not affect x -> not connected
  friend bool connected(sn x, sn y) { return lca(x,y); }
  // # nodes above
  int distRoot() { access(); return getSz(c[0]); }
  sn getRoot() { // get root of LCT component
    access(); sn a = this;
    while (a->c[0]) a = a->c[0], a->prop();
    a->access(); return a;
  }
  sn getPar(int b) { // get b-th parent on path to root
    access(); b = getSz(c[0])-b; assert(b >= 0);
    return fbo(b);
  } // can also get min, max on path to root, etc
  void set(int v) { access(); val = v; calc(); }
  friend void link(sn x, sn y, bool force = 0) {
    assert(!connected(x,y));
    if (force) y->makeRoot(); // make x par of y
    else { y->access(); assert(!y->c[0]); }
    x->access(); setLink(y,x,0); y->calc();
  }
  friend void cut(sn y) { // cut y from its parent
    y->access(); assert(y->c[0]);
    y->c[0]->p = NULL; y->c[0] = NULL; y->calc(); }
  friend void cut(sn x, sn y) { // if x, y adj in tree
    x->makeRoot(); y->access();
    assert(y->c[0] == x && !x->c[0] && !x->c[1]); cut(y); }
};
```

ordered-set.hpp

12 lines

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ordered_set;

/*
example: ordered_set s; s.insert(47);
s.order_of_key(k); - returns number of elements less then k
s.find_by_order(k); - returns iterator to k-th element or s.end()
```

s.count() does not exist.
**s.end() doesn't trigger runtime error. returns 0 if compiled using f8*
**/*

sparse-table.hpp

Description: Sparse table for minimum on the range [l, r).

ab1869, 37 lines

```
struct SparseTable
{
  VI t[LOG];
  VI lg;
  int n;

  void init(int _n)
  {
    n = _n;
    lg.resize(n + 1);
    FOR(i, 2, n + 1)
      lg[i] = lg[i / 2] + 1;

    FOR(j, 0, LOG)
      t[j].assign(n, INF);
  }

  void build(const VI& v)
  {
    FOR (i, 0, SZ(v)) t[0][i] = v[i];

    FOR (j, 1, LOG)
    {
      int len = 1 << (j - 1);
      FOR (i, 0, n - (1 << j) + 1)
      {
        t[j][i] = min(t[j - 1][i], t[j - 1][i + len]);
      }
    }
  }
  // [l, r)
  int query(int l, int r)
  {
    int i = lg[r - l];
    return min(t[i][l], t[i][r - (1 << i)]);
  }
};
```

convex-hull-trick.hpp

Description: add(a, b) adds a straight line $y = ax + b$. getMaxY(p) finds the maximum y at $x = p$.

bb0dd6, 72 lines

```
struct Line
{
  LL a, b, xLast;
  Line() {}
  Line(LL _a, LL _b): a(_a), b(_b) {}
  bool operator<(const Line& l) const
  {
    return MP(a, b) < MP(l.a, l.b);
  }
  bool operator<(int x) const
  {
    return xLast < x;
  }
  __int128 getY(__int128 x) const
  {
    return a * x + b;
  }
  LL intersect(const Line& l) const
  {
    assert(a < l.a);
    LL dA = l.a - a, dB = b - l.b, x = dB / dA;
    if (dB < 0 && dB % dA != 0)
      x--;
    return x;
  }
};

struct ConvexHull: set<Line, less<>>
{
  bool needErase(iterator it, const Line& l)
  {
    LL x = it->xLast;
    if (it->getY(x) > l.getY(x))
      return false;
    if (it == begin())
      return it->a >= l.a;
    x = prev(it)->xLast + 1;
    return it->getY(x) < l.getY(x);
  }
  void add(LL a, LL b)
  {
    Line l(a, b);
    auto it = lower_bound(l);
    if (it != end())
    {
      LL x = it == begin() ? -LINF :
          prev(it)->xLast;
      if ((it == begin()
        || prev(it)->getY(x) >= l.getY(x))
        && it->getY(x + 1) >= l.getY(x + 1))
        return;
    }
    while (it != end() && needErase(it, l))
      it = erase(it);
    while (it != begin() && needErase(prev(it), l))
      erase(prev(it));
    if (it != begin())
    {
      auto itP = prev(it);
      Line itL = *itP;
      itL.xLast = itP->intersect(l);
      erase(itP);
      insert(itL);
    }
    l.xLast = it == end() ? LINF : l.intersect(*it);
    insert(l);
  }
  LL getMaxY(LL p)
  {
    return lower_bound(p)->getY(p);
  }
};
```

Graphs (3)

Decompositions

centroid.hpp

19ecf3, 51 lines

```
VI g[N];
int sz[N];
bool usedC[N];

int dfsSZ(int v, int par)
{
  sz[v] = 1;
  for (auto to : g[v])
  {
    if (to != par && !usedC[to])
      sz[v] += dfsSZ(to, v);
  }
  return sz[v];
}

void build(int u)
{
  dfsSZ(u, -1);
  int szAll = sz[u];
  int pr = u;
  while (true)
  {
    int v = -1;
    for (auto to : g[u])
    {
      if (to == pr || usedC[to])
        continue;
      if (sz[to] * 2 > szAll)
      {
        v = to;
        break;
      }
    }
    if (v == -1)
      break;
    pr = u;
    u = v;
  }
  int cent = u;
  usedC[cent] = true;

  // here calculate f(cent)

  for (auto to : g[cent])
  {
    if (!usedC[to])
    {
      build(to);
    }
  }
}
```

hld.hpp

Description: Run dfsSZ(root, -1, 0) and dfsHLD(root, -1, root) to build the HLD. Each vertex v has an index tin$[v]$. To update on the path, use the process as defined in get(). The values are stored in the vertices.

40c18a, 67 lines

```
VI g[N];
int sz[N];
```

```
int h[N];
int p[N];
int top[N];
int tin[N];
int tout[N];
int t = 0;

void dfsSZ(int v, int par, int hei)
{
  sz[v] = 1;
  h[v] = hei;
  p[v] = par;
  for (auto& to : g[v])
  {
    if (to == par)
      continue;
    dfsSZ(to, v, hei + 1);
    sz[v] += sz[to];
    if (g[v][0] == par || sz[g[v][0]] < sz[to])
      swap(g[v][0], to);
  }
}
void dfsHLD(int v, int par, int tp)
{
  tin[v] = t++;
  top[v] = tp;
  FOR (i, 0, SZ(g[v]))
  {
    int to = g[v][i];
    if (to == par)
      continue;
    if (i == 0)
      dfsHLD(to, v, tp);
    else
      dfsHLD(to, v, to);
  }
  tout[v] = t - 1;
}
LL get(int u, int v)
{
  LL res = 0;
  while(true)
  {
    int tu = top[u];
    int tv = top[v];
    if (tu == tv)
    {
      int t1 = tin[u];
      int t2 = tin[v];
      if (t1 > t2)
        swap(t1, t2);
      // query [t1, t2] both inclusive
      res += query(t1, t2);
      break;
    }
    if (h[tu] < h[tv])
    {
      swap(tu, tv);
      swap(u, v);
    }
    res += query(tin[tu], tin[u]);
    u = p[tu];
  }
  return res;
}
```

biconnected-components.hpp

Description: Colors the edges so that the vertices, connected with the same color are still connected if you delete any vertex.
Time: $\mathcal{O}(m)$

18956b, 137 lines

```
struct Graph
{
  vector<PII> edges;
  vector<VI> g;

  VI used, par;
  VI tin, low, inComp;
  int t = 0, c = 0;
  VI st;

  // components of vertices
  // a vertex can be in several components
  vector<VI> verticesCol;
  // components of edges
  vector<VI> components;
  // col[i] - component of the i-th edge
  VI col;

  int n, m;

  // don't reuse
  void init(int _n, int _m)
  {
    n = _n;
    m = _m;

    edges.assign(m, {0, 0});
    g.assign(n, {});

    used.assign(n, false);
    par.assign(n, -1);

    tin.assign(n, 0);
    low.assign(n, 0);
    inComp.assign(n, 0);

    t = c = 0;

    components.clear();
    col.assign(m, -1);
  }

  void addEdge(int a, int b, int i)
  {
    assert(0 <= a && a < n);
    assert(0 <= b && b < n);
    assert(0 <= i && i < m);

    edges[i] = MP(a, b);
    g[a].PB(i);
    g[b].PB(i);
  }

  void addComp()
  {
    unordered_set<int> s;
    s.reserve(7 * SZ(components[c]));
    for (auto e : components[c])
    {
      s.insert(edges[e].F);
      s.insert(edges[e].S);
      inComp[edges[e].F] = true;
      inComp[edges[e].S] = true;
    }
    verticesCol.PB(VI(ALL(s)));
  }

  void dfs(int v, int p = -1)
  {
    used[v] = 1;
    par[v] = p;
    low[v] = tin[v] = t++;
    int cnt = 0;
    for (auto e : g[v])
    {
      int to = edges[e].F;
      if (to == v)
        to = edges[e].S;

      if (p == to) continue;
      if (!used[to])
      {
        cnt++;
        st.PB(e);
        dfs(to, v);

        low[v] = min(low[v], low[to]);

        if ((par[v] == -1 && cnt > 1) ||
        (par[v] != -1 && low[to] >= tin[v]))
        {
          components.PB({});
          while (st.back() != e)
          {
            components[c].PB(st.back());
            col[st.back()] = c;

            st.pop_back();
          }
          components[c].PB(st.back());
          addComp();
          col[st.back()] = c++;

          st.pop_back();
        }
      }
      else
      {
        low[v] = min(low[v], tin[to]);
        if (tin[to] < tin[v])
          st.PB(e);
      }
    }
```

```
  }
  void build()
  {
    FOR (i, 0, n)
    {
      if (used[i]) continue;
      dfs(i, -1);
      if (st.empty()) continue;
      components.PB({});
      while (!st.empty())
      {
        int e = st.back();
        col[e] = c;
        components[c].PB(e);
        st.pop_back();
      }
      addComp();
      c++;
    }
    FOR (i, 0, n)
      if (!inComp[i])
        verticesCol.PB(VI(1, i));
  }
};
```

Hierholzer's algorithm

hierholzer.hpp

Description: Finds an Eulerian path in a directed or undirected graph. g is a graph with n vertices. $g[u]$ is a vector of pairs $(v, \text{edge\_id})$. m is the number of edges in the graph. The vertices are numbered from 0 to $n-1$, and the edges - from 0 to $m-1$. If there is no Eulerian path, returns $\{\{-1\},\{-1\}\}$. Otherwise, returns the path in the form (vertices, edges) with vertices containing $m+1$ elements and edges containing m elements. If you need an Eulerian cycle, check vertices[0] = vertices.back().

50cce1, 101 lines

```
// 528807 for undirected
tuple<bool, int, int> checkDirected(vector<vector<PII>>& g)
{
  int n = SZ(g), v1 = -1, v2 = -1;
  bool bad = false;
  VI degIn(n);
  FOR(u, 0, n)
    for (auto [v, e] : g[u])
      degIn[v]++;
  FOR(u, 0, n)
  {
    bad |= abs(degIn[u] - SZ(g[u])) > 1;
    if (degIn[u] < SZ(g[u]))
    {
      bad |= v2 != -1;
      v2 = u;
    }
    else if (degIn[u] > SZ(g[u]))
    {
      bad |= v1 != -1;
      v1 = u;
    }
  }
  return {bad, v1, v2};
}

/*tuple<bool, int, int> checkUndirected(vector<vector<PII>>& g)
{
  int n = SZ(g), v1 = -1, v2 = -1;
  bool bad = false;
  FOR(u, 0, n)
  {
    if (SZ(g[u]) & 1)
    {
      bad |= v2 != -1;
      if (v1 == -1)
        v1 = u;
      else
        v2 = u;
    }
  }
  return {bad, v1, v2};
}*/

pair<VI, VI> hierholzer(vector<vector<PII>> g, int m)
{
  // checkUndirected if undirected
  auto [bad, v1, v2] = checkDirected(g);
  if (bad)
    return {{-1}, {-1}};
  if (v1 != -1)
  {
    g[v1].PB({v2, m});
    // uncomment if undirected
    //g[v2].PB({v1, m});
    m++;
  }
  deque<PII> d;
  VI used(m);
  int v = 0, k = 0;
  while (m > 0 && g[v].empty())
    v++;
  while (SZ(d) < m)
  {
    while (k < m)
    {
      while (!g[v].empty() && used[g[v].back().S])
        g[v].pop_back();
      if (!g[v].empty())
        break;
      d.push_front(d.back());
      d.pop_back();
      v = d.back().F;
      k++;
    }
    if (k == m)
      return {{-1}, {-1}};
    d.PB(g[v].back());
    used[g[v].back().S] = true;
    g[v].pop_back();
    v = d.back().F;
  }
  while (v1 != -1 && d.back().S != m - 1)
  {
    d.push_front(d.back());
    d.pop_back();
    v = d.back().F;
  }
  VI vertices = {v}, edges;
  for (auto [u, e] : d)
  {
    vertices.PB(u);
    edges.PB(e);
  }
  if (v1 != -1)
  {
    vertices.pop_back();
    edges.pop_back();
  }
  return {vertices, edges};
}
```

Maximum matching

kuhn.hpp

Description: mateFor is -1 or mate. addEdge($[0, L)$, $[0, R)$).
Time: 0.6s for $L, R \leq 10^5, |E| \leq 2 \cdot 10^5$

bafa1a, 81 lines

```
struct Graph
{
  int szL, szR;
  // edges from the left to the right, 0-indexed
  vector<VI> g;
  VI mateForR, mateForL, usedL;

  void init(int L, int R)
  {
    szL = L, szR = R;
    g.resize(szL);
    mateForL.resize(szL);
    usedL.resize(szL);

    mateForR.resize(szR);
  }

  void addEdge(int from, int to)
  {
    assert(0 <= from && from < szL);
    assert(0 <= to && to < szR);

    g[from].PB(to);
  }

  int iter;
  bool kuhn(int v)
  {
    if (usedL[v] == iter) return false;
    usedL[v] = iter;
    shuffle(ALL(g[v]), rng);
    for(int to : g[v])
    {
      if (mateForR[to] == -1)
      {
        mateForR[to] = v;
        mateForL[v] = to;
        return true;
      }
    }
    for(int to : g[v])
    {
      if (kuhn(mateForR[to]))
```

```
      {
        mateForR[to] = v;
        mateForL[v] = to;
        return true;
      }
    }
    return false;
  }
  int doKuhn()
  {
    fill(ALL(mateForR), -1);
    fill(ALL(mateForL), -1);
    fill(ALL(usedL), -1);

    int res = 0;
    iter = 0;

    while(true)
    {
      iter++;

      bool ok = false;
      FOR(v, 0, szL)
      {
        if (mateForL[v] == -1)
        {
          if (kuhn(v))
          {
            ok = true;
            res++;
          }
        }
      }
      if (!ok) break;
    }
    return res;
  }
};
```

edmonds-blossom.hpp

Description: Finds the maximum matching in a graph.
Time: $\mathcal{O}\left(n^2 m\right)$

490491, 131 lines

```
struct Graph
{
  int n;
  vector<VI> g;
  VI label, first, mate;

  void init(int _n)
  {
    n = _n;
    g.clear();
    g.resize(n + 1);
    label.resize(n + 1);
    first.resize(n + 1);
    mate.resize(n + 1, 0);
  }
  void addEdge(int u, int v)
  {
    assert(0 <= u && u < n);
    assert(0 <= v && v < n);
    u++;
    v++;
    g[u].PB(v);
    g[v].PB(u);
  }
  void augmentPath(int v, int w)
  {
    int t = mate[v];
    mate[v] = w;
    if (mate[t] != v)
      return;
    if (label[v] <= n)
    {
      mate[t] = label[v];
      augmentPath(label[v], t);
      return;
    }
    int x = label[v] / (n + 1);
    int y = label[v] % (n + 1);
    augmentPath(x, y);
    augmentPath(y, x);
  }
  int findMaxMatching()
  {
    FOR(i, 0, n + 1)
      assert(mate[i] == 0);
    int mt = 0;
    DSU dsu;
    FOR(u, 1, n + 1)
    {
      if (mate[u] != 0)
        continue;
      fill(ALL(label), -1);
      iota(ALL(first), 0);
      dsu.init(n + 1);
      label[u] = 0;
      dsu.unite(u, 0);
      queue<int> q;
      q.push(u);
      while (!q.empty())
      {
        int x = q.front();
        q.pop();
        for (int y: g[x])
        {
          if (mate[y] == 0 && y != u)
          {
            mate[y] = x;
            augmentPath(x, y);
            while (!q.empty())
              q.pop();
            mt++;
            break;
          }
          if (label[y] < 0)
          {
            int v = mate[y];
            if (label[v] < 0)
            {
              label[v] = x;
              dsu.unite(v, y);
              q.push(v);
            }
          }
          else
          {
            int r = first[dsu.find(x)], s = first[dsu.find(y)];
            if (r == s)
              continue;
            int edgeLabel = (n + 1) * x + y;
            label[r] = label[s] = -edgeLabel;
            int join;
            while (true)
            {
              if (s != 0)
                swap(r, s);
              r = first[dsu.find(label[mate[r]])];
              if (label[r] == -edgeLabel)
              {
                join = r;
                break;
              }
              label[r] = -edgeLabel;
            }
            for (int z: {x, y})
            {
              for (int v = first[dsu.find(z)];
                v != join;
                v = first[dsu.find(label[mate[v]])])
              {
                label[v] = edgeLabel;
                if (dsu.unite(v, join))
                  first[dsu.find(join)] = join;
                q.push(v);
              }
            }
          }
        }
      }
    }
    return mt;
  }
  int getMate(int v)
  {
    assert(0 <= v && v < n);
    v++;
    int u = mate[v];
    assert(u == 0 || mate[u] == v);
    u--;
    return u;
  }
};
```

Tutte matrix

Given an undirected graph $G = (V, E)$, its Tutte matrix is:

$$T_{ij} = \begin{cases} x_{ij} & \text{if } i < j \text{ and } (i,j) \in E \\ -x_{ji} & \text{if } i > j \ (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

$\det(T) \neq 0$ if and only if G has a perfect matching.

Flows

dinic.hpp

Description: Finds the maximum flow in a network.
Time: $\mathcal{O}\left(n^2 m\right)$.

86349e, 95 lines

```
struct Graph
{
  struct Edge
  {
    int from, to;
    LL cap, flow;
  };

  int n;
  vector<Edge> edges;
  vector<VI> g;
  VI d, p;

  void init(int _n)
  {
    n = _n;
    edges.clear();
    g.clear();
    g.resize(n);
    d.resize(n);
    p.resize(n);
  }
  void addEdge(int from, int to, LL cap)
  {
    assert(0 <= from && from < n);
    assert(0 <= to && to < n);
    assert(0 <= cap);
    g[from].PB(SZ(edges));
    edges.PB({from, to, cap, 0});
    g[to].PB(SZ(edges));
    edges.PB({to, from, 0, 0});
  }
  int bfs(int s, int t)
  {
    fill(ALL(d), -1);
    d[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty())
    {
      int v = q.front();
      q.pop();
      for (int e : g[v])
      {
        int to = edges[e].to;
        if (edges[e].flow < edges[e].cap && d[to] == -1)
        {
          d[to] = d[v] + 1;
          q.push(to);
        }
      }
    }
    return d[t];
  }
  LL dfs(int v, int t, LL flow)
  {
    if (v == t || flow == 0)
      return flow;
    for (; p[v] < SZ(g[v]); p[v]++)
    {
      int e = g[v][p[v]], to = edges[e].to;
      LL c = edges[e].cap, f = edges[e].flow;
      if (f < c && (to == t || d[to] == d[v] + 1))
      {
        LL push = dfs(to, t, min(flow, c - f));
        if (push > 0)
        {
          edges[e].flow += push;
          edges[e ^ 1].flow -= push;
          return push;
        }
      }
    }
    return 0;
  }
  LL flow(int s, int t)
  {
    assert(0 <= s && s < n);
    assert(0 <= t && t < n);
    assert(s != t);
    LL flow = 0;
    while (bfs(s, t) != -1)
    {
      fill(ALL(p), 0);
      while (true)
      {
        LL f = dfs(s, t, LINF);
        if (f == 0)
          break;
        flow += f;
      }
    }
    return flow;
  }
};
```

successive-shortest-path.hpp

Description: Finds the minimum cost maximum flow in a network.
Time: $\mathcal{O}\left(|F| \cdot m \log n\right)$

8af07b, 96 lines

```
struct Graph
{
  struct Edge
  {
    int from, to;
    int cap, flow;
    LL cost;
  };

  int n;
  vector<Edge> edges;
  vector<VI> g;
  vector<LL> pi, d;
  VI pred;
  void init(int _n)
  {
    n = _n;
    edges.clear();
    g.clear();
    g.resize(n);
    pi.assign(n, 0);
    d.resize(n);
    pred.resize(n);
  }
  void addEdge(int from, int to, int cap, LL cost)
  {
    assert(0 <= from && from < n);
    assert(0 <= to && to < n);
    assert(0 <= cap);
    assert(0 <= cost);
    g[from].PB(SZ(edges));
    edges.PB({from, to, cap, 0, cost});
    g[to].PB(SZ(edges));
    edges.PB({to, from, 0, 0, -cost});
  }
  pair<int, LL> flow(int s, int t)
  {
    assert(0 <= s && s < n);
    assert(0 <= t && t < n);
    assert(s != t);
    int flow = 0;
    LL cost = 0;
    while (true)
    {
      fill(ALL(d), LINF);
      fill(ALL(pred), -1);
      d[s] = 0;
      priority_queue<pair<LL, int>> q;
      q.push({0, s});
      while (!q.empty())
      {
        auto [dv, v] = q.top();
        q.pop();
        if (v == t)
          break;
        if (-dv != d[v])
          continue;
        for (int i : g[v])
        {
          if (edges[i].flow == edges[i].cap)
            continue;
          int to = edges[i].to;
          LL nd = d[v] + edges[i].cost + pi[v] - pi[to];
          if (nd < d[to])
          {
            d[to] = nd;
            pred[to] = i;
            q.push({-nd, to});
          }
        }
      }
      if (d[t] == LINF)
        break;
      int curFlow = INF;
      for (int v = t; v != s;)
      {
        int i = pred[v];
        curFlow = min(curFlow, edges[i].cap - edges[i].flow);
        v = edges[i].from;
      }
```

```
      for (int v = t; v != s;)
      {
        int i = pred[v];
        edges[i].flow += curFlow;
        edges[i ^ 1].flow -= curFlow;
        v = edges[i].from;
      }
      flow += curFlow;
      cost += (d[t] + pi[t] - pi[s]) * curFlow;
      FOR(u, 0, n)
        if (d[u] <= d[t])
          pi[u] += d[u] - d[t];
    }
    return {flow, cost};
  }
};
```

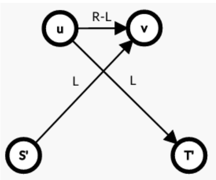
Maximum flow with minimum capacities

`https://atcoder.jp/contests/abc285/editorial/5535`

On the resulting graph, accumulate maximum flow in the following order:

- from S' to T'
- from S' to T
- from S to T'
- from S to T.

An $S - T$ flow that satisfies the minimum capacities exists if and only if, for all outgoing edges from S' and incoming edges to T', the flow and capacity are equal.



Quadratic supermodular pseudoboolean optimization

$$\sum_i a_i x_i + \sum_i b_i \overline{x_i} + \sum_{i,j} c_{ij} x_i \overline{x_j} \to \min$$

$$c_{ij} x_i x_j = c_{ij} x_i - c_{ij} x_i \overline{x_j}$$

If $a_i \le b_i$, add an edge from S to i of capacity $b_i - a_i$ and add a_i to the answer.

Otherwise, add an edge from i to T of capacity $a_i - b_i$ and add b_i to the answer.

Add an edge from i to j of capacity c_{ij}.

Add the $S - T$ minimum cut to the answer.

Matching tricks

Minimum cut

To find the min-cut, search from vertex S on unsaturated edges. Original edges from used vertices to unused ones are in the min-cut.

Minimum vertex cover

The vertex cover problem is not NP-complete in bipartite graphs. The minimum number of vertices required to cover all **edges** is equal to the size of the maximum matching. To reconstruct the minimum vertex cover, create a directed graph:

- matched edges from the right part to the left part
- unmatched edges from the left part to the right part.

Start traversal from unmatched vertices in the left part. The cover includes vertices from the matching:

- unvisited vertices in the left part
- visited vertices in the right part.

Maximum independent set

The independent set problem is not NP-complete in bipartite graphs. It is the complement of the minimum vertex cover.

Minimum edge cover

A minimum edge cover can be found in **any** graph. The minimum number of edges required to cover all vertices can only be determined in graphs without isolated vertices. By utilizing one edge in the matching, we cover two vertices, while any other vertices are covered using one edge for each.

DAG paths

In a DAG, you can find the minimum number of non-intersecting paths that cover all vertices. Duplicate vertices and create a bipartite graph with edges $u_L \to v_R$. Edges in the matching correspond to edges in the paths.

Dominating set

A dominating set for a graph is a subset D of V such that any vertex is in D, or has a neighbor in D. The dominating set problem is NP-complete **even on bipartite graphs**. It can be found greedily on a tree.

Dominator tree

dominator-tree.hpp

Description: Works for cyclic graphs. par – parent in dfs. p – parent in the DSU. val – vertex with the minimum sdom in dsu. dom – immediate dominator. sdom – semidominator, min vertex with alternate path. bkt – vertices with this sdom. dom[root] = -1. dom[v] = -1 if v is unreachable.
Time: $\mathcal{O}(n)$

c9472a, 117 lines

```
struct Graph
{
  int n;
  vector<VI> g, gr, bkt;
  VI par, used, p, val, sdom, dom, tin;
  int T;
  VI ord;

  void init(int _n)
  {
    n = _n;
    g.resize(n);
    gr.resize(n);
    bkt.resize(n);
    par.resize(n);
    used.resize(n);
    p.resize(n);
    val.resize(n);
    sdom.resize(n);
    dom.resize(n);
    tin.resize(n);
  }

  void addEdge(int u, int v)
  {
    g[u].PB(v);
    gr[v].PB(u);
  }

  int find(int v)
  {
    if (p[v] == v)
      return v;
    int y = find(p[v]);
    if (p[y] == y)
      return v;
    if (tin[sdom[val[p[v]]]] < tin[sdom[val[v]]])
      val[v] = val[p[v]];
    p[v] = y;
    return y;
  }
  int get(int v)
  {
    find(v);
    // return vertex with min sdom
    return val[v];
  }

  void dfs(int v, int pr)
  {
    tin[v] = T++;
    used[v] = true;
    ord.PB(v);
    par[v] = pr;
    for (auto to : g[v])
    {
      if (!used[to])
        dfs(to, v);
    }
  }

  void build(int s)
  {
    FOR (i, 0, n)
```

```
    {
      used[i] = false;
      sdom[i] = i;
      dom[i] = -1;
      p[i] = i;
      val[i] = i;
      bkt[i].clear();
    }
    ord.clear();
    T = 0;

    dfs(s, -1);

    RFOR(i, SZ(ord), 0)
    {
      int v = ord[i];
      for (auto from : gr[v])
      {
        // don't consider unreachable vertices
        if (!used[from])
          continue;
        // find min sdom
        if (tin[sdom[v]] > tin[sdom[get(from)]])
        {
          sdom[v] = sdom[get(from)];
        }
      }
      if (v != s)
        bkt[sdom[v]].PB(v);
      for (auto y : bkt[v])
      {
        int u = get(y);
        // if sdoms equals then this is dom
        // else we will find it later
        if (sdom[y] == sdom[u])
          dom[y] = sdom[y];
        else dom[y] = u;
      }
      // add vertex to dsu
      if (par[v] != -1)
        p[v] = par[v];
    }

    for (auto v : ord)
    {
      if (v == s || dom[v] == -1)
        continue;
      if (dom[v] != sdom[v]) dom[v] = dom[dom[v]];
    }
  }

};
```

Sqrt problems

3-cycles.hpp

Description: Finds all triangles in a graph. Each triangle (v, u, w) increments the cnt.
Time: $\mathcal{O}\left(m \cdot \sqrt{m}\right)$

61be84, 42 lines

```
int triangles(int n, vector<PII> edges)
{
  vector<VI> g(n);
  int m = SZ(edges);
  VI deg(n, 0);
  FOR(i, 0, m)
  {
    auto [u, v] = edges[i];
    assert(0 <= u && u < n);
    assert(0 <= v && v < n);
    deg[u]++;
    deg[v]++;
  }
  FOR (i, 0, m)
  {
    auto [u, v] = edges[i];
    if (MP(deg[u], u) < MP(deg[v], v))
      g[u].PB(v);
    else
      g[v].PB(u);
  }
  int cnt = 0;
  VI used(n, 0);
  FOR (v, 0, n)
  {
    for (auto u : g[v])
      used[u] = 1;
    for (auto u : g[v])
    {
      for(auto w : g[u])
      {
        if (used[w])
        {
          cnt++;
        }
      }
    }
    for (auto u : g[v])
      used[u] = 0;
  }
  return cnt;
}
```

4-cycles.hpp

Description: Sort d and add breaks to speed up. With breaks works 0.5s for m $= 5 \cdot 10^5$.
Time: $\mathcal{O}\left(\sum_{uv \in E} \min(\deg(u), \deg(v))\right) = \mathcal{O}\left(m \cdot \sqrt{m}\right)$

e2c43b, 20 lines

```
LL rect(int n)
{
  LL cnt4 = 0;
  vector<PII> d(n);
  FOR (v, 0, n) d[v] = MP(SZ(g[v]), v);
  VI L(n);
  FOR (v, 0, n)
  {
    for (auto u : g[v])
      if (d[u] < d[v])
        for (auto y : g[u])
          if (d[y] < d[v])
            cnt4 += L[y], L[y]++;
    for (auto u : g[v])
      if (d[u] < d[v])
        for (auto y : g[u])
          L[y] = 0;
  }
  return cnt4;
}
```

Shortest path problem for graphs with Monge edge weights

monge-shortest-path.hpp

Description: Finds shortest paths from the vertex 0 to all vertices in a DAG with n vertices, where the edges weights $c(i, j)$ satisfy the Monge property: $\forall i, j, k, l, \quad 0 \le i < j < k < l < n \implies c(i, l) + c(j, k) \ge c(i, k) + c(j, l)$.
Time: $\mathcal{O}\left(n \log n\right)$

0c3fe3, 34 lines

```
template<typename F>
vector<LL> mongeShortestPath(int n, const F& cost)
{
  vector<LL> dist(n, LINF);
  VI amin(n);
  dist[0] = 0;

  auto update = [&](int i, int k)
  {
    LL nd = dist[k] + cost(k, i);
    if (nd < dist[i])
    {
      dist[i] = nd;
      amin[i] = k;
    }
  };

  function<void(int, int)> solve = [&](int l, int r)
  {
    if (r - l == 1)
      return;
    int m = (l + r) / 2;
    FOR(k, amin[l], min(m, amin[r] + 1))
      update(m, k);
    solve(l, m);
    FOR(k, l + 1, m + 1)
      update(r, k);
    solve(m, r);
  };

  update(n - 1, 0);
  solve(0, n - 1);
  return dist;
}
```

Strings (4)

aho-corasick.hpp

c51141, 67 lines

```
const int AL = 26;

struct Node
{
  int p;
  int c;
  int g[AL];
  int nxt[AL];
  int link;
```

```
  Node(int _c, int _p)
  {
    c = _c;
    p = _p;
    fill(g, g + AL, -1);
    fill(nxt, nxt + AL, -1);
    link = -1;
  }
};

struct AC
{
  vector<Node> a;
  void init(int n)
  {
    a.reserve(n);
    a.PB(Node(-1, -1));
  }
  int addStr(const string& s)
  {
    int v = 0;
    FOR (i, 0, SZ(s))
    {
      // change to [0 AL)
      int c = s[i] - 'a';
      if (a[v].nxt[c] == -1)
      {
        a[v].nxt[c] = SZ(a);
        a.PB(Node(c, v));
      }
      v = a[v].nxt[c];
    }
    return v;
  }
  int go(int v, int c)
  {
    if (a[v].g[c] != -1)
      return a[v].g[c];

    if (a[v].nxt[c] != -1)
      a[v].g[c] = a[v].nxt[c];
    else if (v != 0)
      a[v].g[c] = go(getLink(v), c);
    else
      a[v].g[c] = 0;

    return a[v].g[c];
  }
  int getLink(int v)
  {
    if (a[v].link != -1)
      return a[v].link;
    if (v == 0 || a[v].p == 0)
      return 0;
    return a[v].link = go(getLink(a[v].p), a[v].c);
  }
};
```

automaton.hpp

8c531f, 65 lines

```
const int AL = 26;
struct Node
{
  int g[AL];
  int link;
  int len;
  int cnt;
  Node()
  {
    fill(g, g + AL, -1);
    link = -1;
    len = -1;
    cnt = 1;
  }
};

struct Automaton
{
  vector<Node> a;
  int head;
  void init(int n)
  {
    a.reserve(2 * n);
    a.PB(Node());
    head = 0;
  }
  void add(char c)
  {
    // change to [0 AL)
    int ch = c - 'a';
    int nhead = SZ(a);
    a.PB(Node());
    a[nhead].len = a[head].len + 1;
    int cur = head;
    head = nhead;
    while (cur != -1 && a[cur].g[ch] == -1)
    {
      a[cur].g[ch] = head;
      cur = a[cur].link;
    }
    if (cur == -1)
    {
      a[head].link = 0;
      return;
    }
    int p = a[cur].g[ch];
    if (a[p].len == a[cur].len + 1)
    {
      a[head].link = p;
      return;
    }
    int q = SZ(a);
    a.PB(Node());
    a[q] = a[p];
    a[q].cnt = 0;
    a[q].len = a[cur].len + 1;
    a[p].link = a[head].link = q;
    while (cur != -1 && a[cur].g[ch] == p)
    {
      a[cur].g[ch] = q;
      cur = a[cur].link;
    }
  }
};
```

suffix-array.hpp

Description: Cast your string to an array. Don't forget about delimiters. No need to add anything at the end. sa represents permutations of positions if you sort all suffixes. $\text{rnk} = \text{sa}^{-1}$.

dd8ab1, 77 lines

```
struct SuffixArray
{
  int n;
  VI s;
  VI sa, rnk;

  void init(const VI& _s)
  {
    n = SZ(_s);
    s = _s;
    sa = suffixArray();
    rnk.resize(n);
    FOR (i, 0, n)
      rnk[sa[i]] = i;
  }

  void countSort(VI& p, const VI& c)
  {
    VI cnt(n);
    FOR (i, 0, n)
      cnt[c[i]]++;
    VI pos(n);
    FOR (i, 1, n)
      pos[i] = pos[i - 1] + cnt[i - 1];
    VI p2(n);
    for (auto x : p)
    {
      int i = c[x];
      p2[pos[i]++] = x;
    }
    p = p2;
  }

  VI suffixArray()
  {
    // strictly smaller than any other element
    s.PB(-INF);
    n++;
    VI p(n), c(n);
    iota(ALL(p), 0);
    sort(ALL(p), [&](int i, int j)
    {
      return s[i] < s[j];
    });
    int x = 0;
    c[p[0]] = 0;
    FOR (i, 1, n)
    {
      if (s[p[i]] != s[p[i - 1]])
        x++;
      c[p[i]] = x;
    }
    int k = 0;
    while ((1 << k) < n)
```

```
    {
      FOR (i, 0, n)
        p[i] = (p[i] - (1 << k) + n) % n;
      countSort(p, c);
      VI c2(n);
      PII pr = {c[p[0]], c[(p[0] + (1 << k)) % n]};
      FOR (i, 1, n)
      {
        PII nx = {c[p[i]], c[(p[i] + (1 << k)) % n]};
        c2[p[i]] = c2[p[i - 1]];
        if (pr != nx)
          c2[p[i]]++;
        pr = nx;
      }
      c = c2;
      k++;
    }
    p.erase(p.begin());
    s.pop_back();
    n--;
    return p;
  }
};
```

lcp.hpp

Description: queryLcp returns the longest common prefix of substrings starting at i and j.

466a2a, 47 lines

```
struct Lcp
{
  VI lcp;
  SuffixArray a;
  SparseTable st;

  void init(const SuffixArray& _a)
  {
    a = _a;
    lcp = lcpArray();
    st.init(SZ(lcp));
    st.build(lcp);
  }

  VI lcpArray()
  {
    lcp.resize(a.n - 1);
    int h = 0;
    FOR (i, 0, a.n)
    {
      if (h > 0)
        h--;
      if (a.rnk[i] == 0)
        continue;
      int j = a.sa[a.rnk[i] - 1];
      for (; j + h < a.n && i + h < a.n; h++)
      {
        if (a.s[j + h] != a.s[i + h])
          break;
      }
      lcp[a.rnk[i] - 1] = h;
    }
    return lcp;
  }
  int queryLcp(int i, int j)
  {
    if (i == a.n || j == a.n)
      return 0;
    assert(i != j); // return n - i ????
    i = a.rnk[i];
    j = a.rnk[j];
    if (i > j)
      swap(i, j);
    // query [i, j)
    return st.query(i, j);
  }
};
```

z.hpp

e27ac7, 23 lines

```
VI zFunction(const string& s)
{
  int n = SZ(s);
  VI z(n);

  int l = 0;
  int r = 0;
  FOR (i, 1, n)
  {
    z[i] = 0;
    if (i <= r)
      z[i] = min(r - i + 1, z[i - l]);

    while(i + z[i] < n && s[i + z[i]] == s[z[i]])
      z[i]++;
    if(i + z[i] - 1 > r)
    {
      r = i + z[i] - 1;
      l = i;
    }
  }
  return z;
}
```

prefix.hpp

500608, 16 lines

```
VI prefixFunction(const string& s)
{
  int n = SZ(s);
  VI p(n);
  p[0] = 0;
  FOR (i, 1, n)
  {
    int j = p[i - 1];
    while(j != 0 && s[i] != s[j])
      j = p[j - 1];

    if (s[i] == s[j]) j++;
    p[i] = j;
  }
  return p;
}
```

manacher.hpp

Description: $s[i - d0_i, i + d0_i - 1]$, $s[i - d1_i + 1, i + d1_i - 1]$ are palindromes.

e14107, 25 lines

```
vector<VI> manacher(const string& s)
{
  int n = SZ(s);
  vector<VI> d(2);
  FOR (t, 0, 2)
  {
    d[t].resize(n);
    int l = -1;
    int r = -1;
    FOR (i, 0, n)
    {
      if (i <= r)
        d[t][i] = min(r - i + 1, d[t][l + (r - i) + 1 - t]);
      while (i + d[t][i] < n  && i + t - d[t][i] - 1 >= 0
        && s[i + d[t][i]] == s[i + t - d[t][i] - 1])
           d[t][i]++;
      if (i + d[t][i] - t > r)
      {
        r = i + d[t][i] - 1;
        l = i - d[t][i] + t;
      }
    }
  }
  return d;
}
```

palindromic-tree.hpp

2e0b47, 62 lines

```
const int AL = 26;

struct Node
{
  int to[AL];
  int link;
  int len;
  Node(int _link, int _len)
  {
    fill(to, to + AL, -1);
    link = _link;
    len = _len;
  }
};
struct PalTree
{
  string s;
  vector<Node> a;
  int last;

  void init(string t)
  {
    a.clear();
    a.reserve(2 * SZ(t));
    a.PB(Node(-1, -1));
    a.PB(Node(0, 0));
    last = 1;
    s = t;
  }
  void add(int idx)
  {
    // change to [0, AL)
    int ch = s[idx] - 'a';

    int cur = last;
```

```
    while (cur != -1)
    {
      int pos = idx - a[cur].len - 1;
      if (pos >= 0 && s[pos] == s[idx])
        break;
      cur = a[cur].link;
    }
    if (a[cur].to[ch] == -1)
    {
      a[cur].to[ch] = SZ(a);
      int link = a[cur].link;
      while (link != -1)
      {
        int pos = idx - a[link].len - 1;
        if (pos >= 0 && s[pos] == s[idx])
          break;
        link = a[link].link;
      }
      if (link == -1)
        link = 1;
      else
        link = a[link].to[ch];
      a.PB(Node(link, a[cur].len + 2));
    }
    last = a[cur].to[ch];
  }
};
```

Geometry (5)

point.hpp

ff2d7c, 91 lines

```
struct Pt
{
  db x, y;
  Pt operator+(const Pt& p) const
  {
    return {x + p.x, y + p.y};
  }
  Pt operator-(const Pt& p) const
  {
    return {x - p.x, y - p.y};
  }
  Pt operator*(db d) const
  {
    return {x * d, y * d};
  }
  Pt operator/(db d) const
  {
    return {x / d, y / d};
  }
};
db sq(const Pt& p)
{
  return p.x * p.x + p.y * p.y;
}
db abs(const Pt& p)
{
  return sqrt(sq(p));
}
int sgn(db x)
{
  return (EPS < x) - (x < -EPS);
}
// Returns ‘p‘ rotated counter-clockwise by ‘a‘
Pt rot(const Pt& p, db a)
{
  db co = cos(a), si = sin(a);
  return {p.x * co - p.y * si,
    p.x * si + p.y * co};
}
// Returns ‘p‘ rotated counter-clockwise by 90 degrees
Pt perp(const Pt& p)
{
  return {-p.y, p.x};
}
db dot(const Pt& p, const Pt& q)
{
  return p.x * q.x + p.y * q.y;
}
// Returns the angle between ‘p‘ and ‘q‘ in [0, pi]
db angle(const Pt& p, const Pt& q)
{
  return acos(clamp(dot(p, q) / abs(p) /
    abs(q), (db)-1.0, (db)1.0));
}
db cross(const Pt& p, const Pt& q)
{
  return p.x * q.y - p.y * q.x;
}
// Positive if R is on the left side of PQ
// negative on the right side,
// and zero if R is on the line containing PQ
db orient(const Pt& p, const Pt& q, const Pt& r)
{
  return cross(q - p, r - p) / abs(q - p);
}
// Checks if argument of ‘p‘ is in [-pi, 0)
bool half(const Pt& p)
{
  assert(sgn(p.x) != 0 || sgn(p.y) != 0);
  return sgn(p.y) == -1 ||
    (sgn(p.y) == 0 && sgn(p.x) == -1);
}
void polarSortAround(const Pt& o, vector<Pt>& v)
{
  sort(ALL(v), [o](Pt p, Pt q)
  {
    p = p - o;
    q = q - o;
    bool hp = half(p), hq = half(q);
    if (hp != hq)
      return hp < hq;
    int s = sgn(cross(p, q));
    if (s != 0)
      return s == 1;
    return sq(p) < sq(q);
  });
}
ostream& operator<<(ostream& os, const Pt& p)
{
  return os << "(" << p.x << "," << p.y << ")";
}
```

line.hpp

83c9af, 50 lines

```
struct Line
{
  // Equation of the line is dot(n, p) + c = 0
  Pt n;
  db c;
  Line (const Pt& _n, db _c): n(_n), c(_c) {}
  // n is the normal vector to the left of PQ
  Line(const Pt& p, const Pt& q):
    n(perp(q - p)), c(-dot(n, p)) {}
  // The "positive side": dot(n, p) + c > 0
  // The "negative side": dot(n, p) + c < 0
  db side(const Pt& p) const
  {
    return dot(n, p) + c;
  }
  db dist(const Pt& p) const
  {
    return abs(side(p)) / abs(n);
  }
  db sqDist(const Pt& p) const
  {
    return side(p) * side(p) / (db)sq(n);
  }
  Line perpThrough(const Pt& p) const
  {
    return {p, p + n};
  }
  bool cmpProj(const Pt& p, const Pt& q) const
  {
    return sgn(cross(p, n) - cross(q, n)) < 0;
  }
  Pt proj(const Pt& p) const
  {
    return p - n * side(p) / sq(n);
  }
  Pt reflect(const Pt& p) const
  {
    return p - n * 2 * side(p) / sq(n);
  }
};
bool parallel(const Line& l1, const Line& l2)
{
  return sgn(cross(l1.n, l2.n)) == 0;
}
Pt inter(const Line& l1, const Line& l2)
{
  db d = cross(l1.n, l2.n);
  assert(sgn(d) != 0);
  return perp(l2.n * l1.c - l1.n * l2.c) / d;
}
```

segment.hpp

687634, 39 lines

```
// Checks if ‘p‘ is in the disk (the region in a plane
// bounded by a circle) of diameter [ab]
bool inDisk(const Pt& a, const Pt& b, const Pt& p)
{
  return sgn(dot(a - p, b - p)) <= 0;
```

```
}
// Checks if ‘p‘ lies on segment [ab]
bool onSegment(const Pt& a, const Pt& b, const Pt& p)
{
  return sgn(orient(a, b, p)) == 0 && inDisk(a, b, p);
}
// Checks if the segments [ab] and [cd] intersect
// properly (their intersection is one point
// which is not an endpoint of either segment)
bool properInter(const Pt& a, const Pt& b, const Pt& c, const Pt& d)
{
  db oa = orient(c, d, a);
  db ob = orient(c, d, b);
  db oc = orient(a, b, c);
  db od = orient(a, b, d);
  return sgn(oa) * sgn(ob) == -1 && sgn(oc) * sgn(od) == -1;
}
// Returns the distance between [ab] and ‘p‘
db segPt(const Pt& a, const Pt& b, const Pt& p)
{
  Line l(a, b);
  assert(sgn(sq(l.n)) != 0);
  if (l.cmpProj(a, p) && l.cmpProj(p, b))
    return l.dist(p);
  return min(abs(p - a), abs(p - b));
}
// Returns the distance between [ab] and [cd]
db segSeg(const Pt& a, const Pt& b, const Pt& c, const Pt& d)
{
  if (properInter(a, b, c, d))
    return 0;
  return min({segPt(a, b, c), segPt(a, b, d),
      segPt(c, d, a), segPt(c, d, b)});
}
```

polygon.hpp

251907, 67 lines

```
bool isConvex(const vector<Pt>& v)
{
  bool hasPos = false, hasNeg = false;
  int n = SZ(v);
  FOR(i, 0, n)
  {
    int s = sgn(orient(v[i], v[(i + 1) % n], v[(i + 2) % n]));
    hasPos |= s > 0;
    hasNeg |= s < 0;
  }
  return !(hasPos && hasNeg);
}
db areaTriangle(const Pt& a, const Pt& b, const Pt& c)
{
  return abs(cross(b - a, c - a)) / 2.0;
}
db areaPolygon(const vector<Pt>& v)
{
  db area = 0.0;
  int n = SZ(v);
  FOR(i, 0, n)
    area += cross(v[i], v[(i + 1) % n]);
  return abs(area) / 2.0;
}
// Checks if point ‘a‘ is inside the convex
// polygon ‘v‘. Returns true if on the boundary.
// ‘v‘ must not contain duplicated vertices.
// Time: O(log n)
bool inConvexPolygon(const vector<Pt>& v, const Pt& a)
{
  assert(SZ(v) >= 2);
  if (SZ(v) == 2)
    return onSegment(v[0], v[1], a);
  if (sgn(orient(v.back(), v[0], a)) < 0
    || sgn(orient(v[0], v[1], a)) < 0)
    return false;
  int i = lower_bound(v.begin() + 2, v.end(), a,
  [&](const Pt& p, const Pt& q)
  {
    return sgn(orient(v[0], p, q)) > 0;
  }) - v.begin();
  return sgn(orient(v[i - 1], v[i], a)) >= 0;
}
bool above(const Pt& a, const Pt& p)
{
  return sgn(p.y - a.y) >= 0;
}
bool crossesRay(const Pt& a, const Pt& p,
  const Pt& q)
{
  return sgn((above(a, q) - above(a, p))
    * orient(a, p, q)) == 1;
}
// Checks if point ‘a‘ is inside the polygon
// If ‘strict‘, false when ‘a‘ is on the boundary
bool inPolygon(const vector<Pt>& v, const Pt& a, bool strict = true)
{
  int numCrossings = 0;
  int n = SZ(v);
  FOR(i, 0, n)
  {
    if (onSegment(v[i], v[(i + 1) % n], a))
      return !strict;
    numCrossings += crossesRay(a, v[i], v[(i + 1) % n]);
  }
  return numCrossings & 1;
}
```

convex-hull.hpp

4efeb1, 27 lines

```
vector<Pt> convexHull(vector<Pt> v)
{
  if (SZ(v) <= 1)
    return v;
  sort(ALL(v), [](const Pt& p, const Pt& q)
  {
    int dx = sgn(p.x - q.x);
    if (dx != 0)
      return dx < 0;
    return sgn(p.y - q.y) < 0;
  });
  vector<Pt> lower, upper;
  for (const Pt& p : v)
  {
    while (SZ(lower) > 1
      && sgn(orient(lower[SZ(lower) - 2], lower.back(), p)) <= 0)
      lower.pop_back();
    while (SZ(upper) > 1
      && sgn(orient(upper[SZ(upper) - 2], upper.back(), p)) >= 0)
      upper.pop_back();
    lower.PB(p);
    upper.PB(p);
  }
  reverse(ALL(upper));
  lower.insert(lower.end(), next(upper.begin()), prev(upper.end()));
  return lower;
}
```

tangents-to-convex-polygon.hpp

Description: Returns the indices of tangent points from p. p must be strictly outside the polygon.

e35dbb, 38 lines

```
PII tangentsToConvexPolygon(const vector<Pt>& v, const Pt& p)
{
  int n = SZ(v), i = 0;
  if (n == 2)
    return {0, 1};
  while (sgn(orient(p, v[i], v[(i + 1) % n]))
    * sgn(orient(p, v[i], v[(i + n - 1) % n])) > 0)
    i++;
  int s1 = 1, s2 = -1;
  if (sgn(orient(p, v[i], v[(i + 1) % n])) == s1
    || sgn(orient(p, v[i], v[(i + n - 1) % n])) == s2)
    swap(s1, s2);
  PII res;
  int l = i, r = i + n - 1;
  while (r - l > 1)
  {
    int m = (l + r) / 2;
    if (sgn(orient(p, v[i], v[m % n])) != s1
      && sgn(orient(p, v[m % n], v[(m + 1) % n])) != s1)
      l = m;
    else
      r = m;
  }
  res.F = r % n;
  l = i;
  r = i + n - 1;
  while (r - l > 1)
  {
    int m = (l + r) / 2;
    if (sgn(orient(p, v[i], v[m % n])) == s2
      || sgn(orient(p, v[m % n], v[(m + 1) % n])) != s2)
      l = m;
    else
      r = m;
  }
  res.S = r % n;
  return res;
}
```

minkowski-sum.hpp

Description: Returns the Minkowski sum of two convex polygons.

6feb84, 40 lines

```
vector<Pt> minkowskiSum(const vector<Pt>& v1, const vector<Pt>& v2)
{
  if (v1.empty() || v2.empty())
    return {};
  if (SZ(v1) == 1 && SZ(v2) == 1)
```

```
    return {v1[0] + v2[0]};
  auto comp = [](const Pt& p, const Pt& q)
  {
    return sgn(p.x - q.x) < 0
      || (sgn(p.x - q.x) == 0
      && sgn(p.y - q.y) < 0);
  };
  int i1 = min_element(ALL(v1), comp) - v1.begin();
  int i2 = min_element(ALL(v2), comp) - v2.begin();
  vector<Pt> res;
  int n1 = SZ(v1), n2 = SZ(v2),
    j1 = 0, j2 = 0;
  while (j1 < n1 || j2 < n2)
  {
    const Pt& p1 = v1[(i1 + j1) % n1];
    const Pt& q1 = v1[(i1 + j1 + 1) % n1];
    const Pt& p2 = v2[(i2 + j2) % n2];
    const Pt& q2 = v2[(i2 + j2 + 1) % n2];
    if (SZ(res) >= 2 && onSegment(res[SZ(res) - 2], p1 + p2, res.back()))
      res.pop_back();
    res.PB(p1 + p2);
    int s = sgn(cross(q1 - p1, q2 - p2));
    if (j1 < n1 && (j2 == n2 || s > 0
      || (s == 0 && (SZ(res) < 2
      || sgn(dot(res.back()
      - res[SZ(res) - 2],
      q1 + p2 - res.back())) > 0))))
      j1++;
    else
      j2++;
  }
  if (SZ(res) > 2 && onSegment(res[SZ(res) - 2], res[0], res.back()))
    res.pop_back();
  return res;
}
```

halfplane-intersection.hpp

Description: Returns the counter-clockwise ordered vertices of the half-plane intersection. Returns empty if the intersection is empty. Adds a bounding box to ensure a finite area.

5c6d01, 47 lines

```
vector<Pt> hplaneInter(vector<Line> lines)
{
  const db C = 1e9;
  lines.PB({{-C, C}, {-C, -C}});
  lines.PB({{-C, -C}, {C, -C}});
  lines.PB({{C, -C}, {C, C}});
  lines.PB({{C, C}, {-C, C}});
  sort(ALL(lines), [](const Line& l1, const Line& l2)
  {
    bool h1 = half(l1.n), h2 = half(l2.n);
    if (h1 != h2)
      return h1 < h2;
    int p = sgn(cross(l1.n, l2.n));
    if (p != 0)
      return p > 0;
    return sgn(l1.c / abs(l1.n) - l2.c / abs(l2.n)) < 0;
  });
  lines.erase(unique(ALL(lines), parallel), lines.end());
  deque<pair<Line, Pt>> d;
  for (const Line& l : lines)
  {
    while (SZ(d) > 1 && sgn(l.side((d.end() - 1)->S)) < 0)
      d.pop_back();
    while (SZ(d) > 1 && sgn(l.side((d.begin() + 1)->S)) < 0)
      d.pop_front();
    if (!d.empty() && sgn(cross(d.back().F.n, l.n)) <= 0)
      return {};
    if (SZ(d) < 2 || sgn(d.front().F.side(inter(l, d.back().F))) >= 0)
    {
      Pt p;
      if (!d.empty())
      {
        p = inter(l, d.back().F);
        if (!parallel(l, d.front().F))
          d.front().S = inter(l, d.front().F);
      }
      d.PB({l, p});
    }
  }
  vector<Pt> res;
  for (auto [l, p] : d)
  {
    if (res.empty() || sgn(sq(p - res.back())) > 0)
      res.PB(p);
  }
  return res;
}
```

circle.hpp

e4d116, 72 lines

```
// Returns the circumcenter of triangle abc.
// The circumcircle of a triangle is a circle that passes through all
//   three vertices.
Pt circumCenter(const Pt& a, Pt b, Pt c)
{
  b = b - a;
  c = c - a;
  assert(sgn(cross(b, c)) != 0);
  return a + perp(b * sq(c) - c * sq(b)) / cross(b, c) / 2;
}
// Returns circle–line intersection points
vector<Pt> circleLine(const Pt& o, db r, const Line& l)
{
  db h2 = r * r - l.sqDist(o);
  if (sgn(h2) == -1)
    return {};
  Pt p = l.proj(o);
  if (sgn(h2) == 0)
    return {p};
  Pt h = perp(l.n) * sqrt(h2) / abs(l.n);
  return {p - h, p + h};
}
// Returns circle–circle intersection points
vector<Pt> circleCircle(const Pt& o1, db r1, const Pt& o2, db r2)
{
  Pt d = o2 - o1;
  db d2 = sq(d);
  if (sgn(d2) == 0)
  {
    // assuming the circles don't coincide
    assert(sgn(r2 - r1) != 0);
    return {};
  }
  db pd = (d2 + r1 * r1 - r2 * r2) / 2;
  db h2 = r1 * r1 - pd * pd / d2;
  if (sgn(h2) == -1)
    return {};
  Pt p = o1 + d * pd / d2;
  if (sgn(h2) == 0)
    return {p};
  Pt h = perp(d) * sqrt(h2 / d2);
  return {p - h, p + h};
}
// Finds common tangents (outer or inner)
// If there are 2 tangents, returns the pairs of
// tangency points on each circle (p1, p2)
// If there is 1 tangent, the circles are tangent
// to each other at some point p, res contains p
// 4 times, and the tangent line can be found as
// line(o1, p).perpThrough(p)
// The same code can be used to find the tangent
// to a circle through a point by setting r2 to 0
// (in which case 'inner' doesn't matter)
vector<pair<Pt, Pt>> tangents(const Pt& o1,
  db r1, const Pt& o2, db r2, bool inner)
{
  if (inner)
    r2 = -r2;
  Pt d = o2 - o1;
  db dr = r1 - r2, d2 = sq(d), h2 = d2 - dr * dr;
  if (sgn(d2) == 0 || sgn(h2) < 0)
  {
    assert(sgn(h2) != 0);
    return {};
  }
  vector<pair<Pt, Pt>> res;
  for (db sign : {-1, 1})
  {
    Pt v = (d * dr + perp(d) * sqrt(h2) * sign) / d2;
    res.PB({o1 + v * r1, o2 + v * r2});
  }
  return res;
}
```

welzl.hpp

Description: Returns the smallest enclosing circle of points in v
Time: $\mathcal{O}(n)$ (expected)

e33f59, 36 lines

```
pair<Pt, db> welzl(vector<Pt> v)
{
  int n = SZ(v), k = 0, idxes[2];
  mt19937 rng;
  shuffle(ALL(v), rng);
  Pt c = v[0];
  db r = 0;
  while (true)
  {
    FOR(i, k, n)
    {
      if (sgn(abs(v[i] - c) - r) > 0)
      {
        swap(v[i], v[k]);
        if (k == 0)
          c = v[0];
        else if (k == 1)
```

```
        c = (v[0] + v[1]) / 2;
      else
        c = circumCenter(v[0], v[1],  v[2]);
      r = abs(v[0] - c);
      if (k < i)
      {
        if (k < 2)
          idxes[k++] = i;
        shuffle(v.begin() + k, v.begin() + i + 1, rng);
        break;
      }
    }
    while (k > 0 && idxes[k - 1] == i)
      k--;
    if (i == n - 1)
      return {c, r};
  }
 }
}
```

closest-pair.hpp

Description: Returns the distance between the closest points
Time: $\mathcal{O}(n \log n)$

8696b6, 23 lines

```
db closestPair(vector<Pt> v)
{
  sort(ALL(v), [](const Pt& p, const Pt& q)
  {
    return sgn(p.x - q.x) < 0;
  });
  set<pair<db, db>> s;
  int n = SZ(v), ptr = 0;
  db h = 1e18;
  FOR(i, 0, n)
  {
    for (auto it = s.lower_bound(MP(v[i].y - h, v[i].x));
      it != s.end() && sgn(it->F - (v[i].y + h)) <= 0; it++)
    {
      Pt q = {it->S, it->F};
      h = min(h, abs(v[i] - q));
    }
    for (; sgn(v[ptr].x - (v[i].x - h)) <= 0; ptr++)
      s.erase({v[ptr].y, v[ptr].x});
    s.insert({v[i].y, v[i].x});
  }
  return h;
}
```

planar-graph.hpp

Description: Finds faces in a planar graph. Use addVertex() and addEdge() for initializing the graph and addQueryPoint() for initializing the queries. After initialization, call findFaces() before using other functions. getIncidentFaces(i) returns the pair of faces (u, v) (possibly $u = v$) such that the i-th edge lies on the boundary of these faces. getFaceOfQueryPoint(i) returns the face where the i-th query point lies.

629940, 169 lines

```
namespace PlanarGraph
{
struct IndexedPt
{
  Pt p;
  int index;
  bool operator<(const IndexedPt& q) const
  {
    return p.x < q.p.x;
  }
};
struct Edge
{
  // cross(vertices[j].p - vertices[i].p, l.n) > 0
  int i, j;
  Line l;
};
vector<IndexedPt> vertices, queryPoints;
vector<Edge> edges;
struct Comparator
{
  using is_transparent = void;
  static IndexedPt vertex;
  db getY(const Line& l) const
  {
    return -(l.n.x * vertex.p.x + l.c) / l.n.y;
  }
  bool operator()(int i, int j) const
  {
    auto [u1, v1, l1] = edges[i];
    auto [u2, v2, l2] = edges[j];
    if (u1 == vertex.index && u2 == vertex.index)
      return sgn(cross(l1.n, l2.n)) > 0;
    if (v1 == vertex.index && v2 == vertex.index)
      return sgn(cross(l1.n, l2.n)) < 0;
    int dy = sgn(getY(l1) - getY(l2));
    assert(dy != 0);
    return dy < 0;
  }
  bool operator()(int i, const Pt& p) const
  {
    int dy = sgn(getY(edges[i].l) - p.y);
    assert(dy != 0);
    return dy < 0;
  }
} comparator;
IndexedPt Comparator::vertex;
DSU dsu;
VI upperFace, queryAns;

void addVertex(const Pt& p)
{
  vertices.PB({p, SZ(vertices)});
}
void addEdge(int i, int j, const Line& l)
{
  assert(0 <= i && i < SZ(vertices));
  assert(0 <= j && j < SZ(vertices));
  assert(i != j);
  assert(vertices[i].index == i);
  assert(vertices[j].index == j);
  edges.PB({i, j, l});
}
void addEdge(int i, int j)
{
  addEdge(i, j, {vertices[i].p, vertices[j].p});
}
void addQueryPoint(const Pt& p)
{
  queryPoints.PB({p, SZ(queryPoints)});
}
void findFaces()
{
  int n = SZ(vertices), m = SZ(edges);
  const db ROT_ANGLE = 4;
  for (auto& p : vertices)
    p.p = rot(p.p, ROT_ANGLE);
  for (auto& p : queryPoints)
    p.p = rot(p.p, ROT_ANGLE);
  vector<VI> edgesL(n), edgesR(n);
  FOR(k, 0, m)
  {
    auto& [i, j, l] = edges[k];
    l.n = rot(l.n, ROT_ANGLE);
    if (vertices[i].p.x > vertices[j].p.x)
    {
      swap(i, j);
      l.n = l.n * (-1);
      l.c *= -1;
    }
    edgesL[j].PB(k);
    edgesR[i].PB(k);
  }
  sort(ALL(vertices));
  sort(ALL(queryPoints));
  // when choosing INF, remember that we rotate the plane
  addVertex({-INF, INF});
  addVertex({INF, INF});
  addEdge(n, n + 1);
  dsu.init(m + 1);
  set<int, Comparator> s;
  s.insert(m);
  upperFace.resize(m);
  int ptr = 0;
  queryAns.resize(SZ(queryPoints));
  for (const IndexedPt& vertex : vertices)
  {
    int i = vertex.index;
    while (ptr < SZ(queryPoints)
      && (i >= n || queryPoints[ptr] < vertex))
    {
      const auto& [pt, j] = queryPoints[ptr++];
      Comparator::vertex = {pt, -1};
      queryAns[j] = *s.lower_bound(pt);
    }
    if (i >= n)
      break;
    Comparator::vertex = vertex;
    int upper = -1, lower = -1;
    if (!edgesL[i].empty())
    {
      sort(ALL(edgesL[i]), comparator);
      auto it = s.lower_bound(edgesL[i][0]);
      lower = edgesL[i][0];
      for (int e : edgesL[i])
      {
        assert(*it == e);
        assert(next(it) != s.end());
        upperFace[e] = *next(it);
```

```
        it = s.erase(it);
      }
      assert(it != s.end());
      upper = *it;
    }
    if (!edgesR[i].empty())
    {
      sort(ALL(edgesR[i]), comparator);
      if (upper == -1)
      {
        upper = *s.lower_bound(edgesR[i][0]);
      }
      int prv = -1;
      for (int e : edgesR[i])
      {
        s.insert(e);
        if (prv != -1)
        {
          upperFace[prv] = e;
        }
        prv = e;
      }
      upperFace[edgesR[i].back()] = upper;
      dsu.unite(edgesL[i].empty() ? upper : lower, edgesR[i][0]);
    }
    else if (lower != -1 && upper != -1)
    {
      dsu.unite(upper, lower);
    }
  }
}
PII getIncidentFaces(int i)
{
  return {dsu.find(i), dsu.find(upperFace[i])};
}
int getFaceOfQueryPoint(int i)
{
  return dsu.find(queryAns[i]);
}
};
```

Mathematics (6)

Number-theoretic algorithms

gcd.hpp

Description: $ax + by = d$, $\gcd(a, b) = |d| \to (d, x, y)$.
Minimizes $|x| + |y|$. And minimizes $|x - y|$ for $a > 0$, $b > 0$.

261b5c, 16 lines

```
tuple<LL, LL, LL> gcdExt(LL a, LL b)
{
  LL x1 = 1, y1 = 0;
  LL x2 = 0, y2 = 1;
  while (b)
  {
    LL k = a / b;
    x1 -= k * x2;
    y1 -= k * y2;
    a %= b;
    swap(a, b);
    swap(x1, x2);
    swap(y1, y2);
  }
  return {a, x1, y1};
}
```

fast-chinese.hpp

Description: $x\%p_i = m_i, \mathrm{lcm}(p_i) \le 10^{18}, 0 \le x < \mathrm{lcm}(p_i) \to x$ or -1.
Time: $\mathcal{O}(n \log(\mathrm{lcm}(p_i)))$

3c13b2, 24 lines

```
LL fastChinese(vector<LL> m, vector<LL> p)
{
  assert(SZ(m) == SZ(p));
  LL aa = p[0];
  LL bb = m[0];
  FOR(i, 1, SZ(m))
  {
    LL b = (m[i] - bb % p[i] + p[i]) % p[i];
    LL a = aa % p[i];
    LL c = p[i];

    auto [d, x, y] = gcdExt(a, c);
    if(b % d != 0)
      return -1;
    a /= d;
    b /= d;
    c /= d;
    b = (b * (__int128)x % c + c) % c;

    bb = aa * b + bb;
    aa = aa * c;
  }
  return bb;
}
```

chinese.hpp

Description: Code finds a specific structure of the answer.
Time: $\mathcal{O}(n^2)$

b8b297, 33 lines

```
LL chinese(VI m, VI p)
{
  int n = SZ(m);
  FOR(i, 1, n)
  {
    LL a = 1;
    LL b = 0;
    RFOR(j, i, 0)
    {
      b = (b * p[j] + m[j]) % p[i];
      a = a * p[j] % p[i];
    }
    b = (m[i] - b + p[i]) % p[i];

    int c = p[i];
    auto [d, x, y] = gcdExt(a, c);

    if(b % d != 0)
      return -1;
    a /= d;
    b /= d;
    c /= d;

    b = (b * x % c + c) % c;
    m[i] = b;
    p[i] = c;
  }
  //specific structure where gcd(pi, pj) = 1
  LL res = m[n - 1];
  RFOR(i, n - 1, 0)
    res = res * p[i] + m[i];
  return res;
}
```

miller-rabin.hpp

Description: To speed up change candidates to at least 4 random values rng() % (n - 3) + 2. **Use __int128 in mult**.
Time: $\mathcal{O}(|\text{candidates}| \cdot \log n)$

394bc8, 33 lines

```
VI candidates = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 47};
bool millerRabin(LL n)
{
  if (n == 1)
    return false;
  if (n == 2 || n == 3)
    return true;
  LL d = n - 1;
  int s = __builtin_ctzll(d);
  d >>= s;

  for (LL b : candidates)
  {
    if (b >= n)
      break;
    b = binpow(b, d, n);
    if (b == 1)
      continue;
    bool ok = false;
    FOR (i, 0, s)
    {
      if (b + 1 == n)
      {
        ok = true;
        break;
      }
      b = mult(b, b, n);
    }
    if (!ok)
      return false;
  }
  return true;
}
```

pollard.hpp

Description: Uses the Miller-Rabin test. rho finds a divisor of n. use __int128 in mult.
Time: $\mathcal{O}\left(n^{1/4} \cdot \log n\right)$.

53da5d, 62 lines

```
LL f(LL x, LL c, LL n)
{
  return add(mult(x, x, n), c, n);
}

LL rho(LL n)
{
  const int iter = 47 * pow(n, 0.25);
  while (true)
```

```
  {
    LL x0 = rng() % n;
    LL c = rng() % n;
    LL x = x0;
    LL y = x0;
    LL g = 1;
    FOR (i, 0, iter)
    {
      x = f(x, c, n);
      y = f(y, c, n);
      y = f(y, c, n);
      g = gcd(abs(x - y), n);
      if (g != 1)
        break;
    }
    if (g > 1 && g < n)
      return g;
  }
}
VI primes = {2, 3, 5, 7, 11, 13, 17, 19, 23};

vector<LL> factorize(LL n)
{
  vector<LL> ans;

  for (auto p : primes)
  {
    while (n % p == 0)
    {
      ans.PB(p);
      n /= p;
    }
  }
  queue<LL> q;
  q.push(n);

  while (!q.empty())
  {
    LL x = q.front();
    q.pop();
    if (x == 1)
      continue;
    if (millerRabin(x))
      ans.PB(x);
    else
    {
      LL y = rho(x);
      q.push(y);
      q.push(x / y);
    }
  }
  return ans;
}
```

floor-sum.hpp

Description: Computes $\sum_{i=0}^{n-1} \left\lfloor \frac{a \cdot i + b}{m} \right\rfloor$.
Time: $\mathcal{O}(\log m)$.

7e1c68, 16 lines

```
LL floorSum(LL n, LL m, LL a, LL b)
{
  LL ans = 0;
  while (true)
  {
    ans += (a / m) * n * (n - 1) / 2 + (b / m) * n;
    a %= m;
    b %= m;
    if (a == 0)
      return ans;
    LL k = (a * (n - 1) + b) / m;
    b = a * n - m * k + b;
    n = k;
    swap(a, m);
  }
}
```

min-mod-linear.hpp

Description: Finds $\min\{(ax+b) \mod m \mid 0 \le x < n\}$.
Time: $\mathcal{O}(\log m)$.

874428, 14 lines

```
int minModLinear(LL n, LL m, LL a, LL b)
{
  LL res = m;
  while (n > 0)
  {
    a %= m;
    b = (b % m + m) % m;
    res = min(res, b);
    n = (a * (n - 1) + b) / m;
    b -= m * n;
    swap(a, m);
  }
  return res;
}
```

mod-inequality.hpp

Description: Finds the smallest $x \ge 0$ such that $(ax+b) \mod m \ge c$. Returns -1, if the solution does not exist.
Time: $\mathcal{O}(\log m)$.

e842a1, 15 lines

```
int modInequality(LL m, LL a, LL b, LL c)
{
  a %= m;
  b %= m;
  if (b >= c)
    return 0;
  if (a == 0)
    return -1;
  if (c + a < m)
    return (c - b + a - 1) / a;
  int k = modInequality(a, m, c - b - 1, c + a - m);
  if (k == -1)
    return -1;
  return (k * m + c - b + a - 1) / a;
}
```

Matrices

gaussian.hpp

Description: Solves the system $Ax = b$. If there is no solution, returns $(\{\}, -1)$. If the solution is unique, returns $(x, 1)$. Otherwise, returns $(x, 2)$ with x being any solution.
Time: $\mathcal{O}(nm \min(n, m))$

12e66c, 50 lines

```
pair<VI, int> solveLinear(vector<VI> a, VI b)
{
  int n = SZ(a), m = SZ(a[0]);
  assert(SZ(b) == n);
  FOR(i, 0, n)
  {
    assert(SZ(a[i]) == m);
    a[i].PB(b[i]);
  }
  int p = 0;
  VI pivots;
  FOR(j, 0, m)
  {
    // with doubles, abs(a[p][j]) -> max
    if (a[p][j] == 0)
    {
      int l = -1;
      FOR(i, p, n)
        if (a[i][j] != 0)
          l = i;
      if (l == -1)
        continue;
      swap(a[p], a[l]);
    }
    int inv = binpow(a[p][j], mod - 2);
    FOR(i, p + 1, n)
    {
      int c = mult(a[i][j], inv);
      FOR(k, j, m + 1)
        updSub(a[i][k], mult(c, a[p][k]));
    }
    pivots.PB(j);
    p++;
    if (p == n)
      break;
  }
  FOR(i, p, n)
    if (a[i].back() != 0)
      return {{}, -1};
  VI x(m);
  RFOR(i, p, 0)
  {
    int j = pivots[i];
    x[j] = a[i].back();
    FOR(k, j + 1, m)
      updSub(x[j], mult(a[i][k], x[k]));
    x[j] = mult(x[j], binpow(a[i][j], mod - 2));
  }
  return {x, SZ(pivots) == m ? 1 : 2};
}
```

hungarian.hpp

Description: Finds a maximum matching that has the minimum weight in a weighted bipartite graph.
Time: $\mathcal{O}(n^2 m)$

ac07d2, 63 lines

```
LL hungarian(const vector<vector<LL>>& a)
{
  int n = SZ(a), m = SZ(a[0]);
  assert(n <= m);
  vector<LL> u(n + 1), v(m + 1);
  VI p(m + 1, n), way(m + 1);
  FOR(i, 0, n)
  {
```

```
    p[m] = i;
    int j0 = m;
    vector<LL> minv(m + 1, LINF);
    VI used(m + 1);
    while (p[j0] != n)
    {
      used[j0] = true;
      int i0 = p[j0], j1 = -1;
      LL delta = LINF;
      FOR(j, 0, m)
      {
        if (!used[j])
        {
          int cur = a[i0][j] - u[i0] - v[j];
          if (cur < minv[j])
          {
            minv[j] = cur;
            way[j] = j0;
          }
          if (minv[j] < delta)
          {
            delta = minv[j];
            j1 = j;
          }
        }
      }
      assert(j1 != -1);
      FOR(j, 0, m + 1)
      {
        if (used[j])
        {
          u[p[j]] += delta;
          v[j] -= delta;
        }
        else
          minv[j] -= delta;
      }
      j0 = j1;
    }
    while (j0 != m)
    {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    }
  }
  VI ans(n + 1);
  FOR(j, 0, m)
    ans[p[j]] = j;
  LL res = 0;
  FOR(i, 0, n)
    res += a[i][ans[i]];
  assert(res == -v[m]);
  return res;
}
```

simplex.hpp

Description: $c^T x \to \max,\ Ax \le b,\ x \ge 0$.

03c648, 142 lines

```
typedef vector<db> VD;

struct Simplex
{
  void pivot(int l, int e)
  {
    assert(0 <= l && l < m);
    assert(0 <= e && e < n);
    assert(abs(a[l][e]) > EPS);
    b[l] /= a[l][e];
    FOR(j, 0, n)
      if (j != e)
        a[l][j] /= a[l][e];
    a[l][e] = 1 / a[l][e];
    FOR(i, 0, m)
    {
      if (i != l)
      {
        b[i] -= a[i][e] * b[l];
        FOR(j, 0, n)
          if (j != e)
            a[i][j] -= a[i][e] * a[l][j];
        a[i][e] *= -a[l][e];
      }
    }
    v += c[e] * b[l];
    FOR(j, 0, n)
      if (j != e)
        c[j] -= c[e] * a[l][j];
    c[e] *= -a[l][e];
    swap(nonBasic[e], basic[l]);
  }
  void findOptimal()
  {
    VD delta(m);
    while (true)
    {
      int e = -1;
      FOR(j, 0, n)
        if (c[j] > EPS && (e == -1 || nonBasic[j] < nonBasic[e]))
          e = j;
      if (e == -1)
        break;
      FOR(i, 0, m)
        delta[i] = a[i][e] > EPS ? b[i] / a[i][e] : LINF;
      int l = min_element(ALL(delta)) - delta.begin();
      if (delta[l] == LINF)
      {
        // unbounded
        assert(false);
      }
      pivot(l, e);
    }
  }
  void initializeSimplex(const vector<VD>& _a, const VD& _b, const VD& _c)
  {
    m = SZ(_b);
    n = SZ(_c);
    nonBasic.resize(n);
    iota(ALL(nonBasic), 0);
    basic.resize(m);
    iota(ALL(basic), n);
    a = _a;
    b = _b;
    c = _c;
    v = 0;
    int k = min_element(ALL(b)) - b.begin();
    if (b[k] > -EPS)
      return;
    nonBasic.PB(n);
    iota(ALL(basic), n + 1);
    FOR(i, 0, m)
      a[i].PB(-1);
    c.assign(n, 0);
    c.PB(-1);
    n++;
    pivot(k, n - 1);
    findOptimal();
    if (v < -EPS)
    {
      // infeasible
      assert(false);
    }
    int l = find(ALL(basic), n - 1) - basic.begin();
    if (l != m)
    {
      int e = -1;
      while (abs(a[l][e]) < EPS)
        e++;
      pivot(l, e);
    }
    n--;
    int p = find(ALL(nonBasic), n) - nonBasic.begin();
    assert(p < n + 1);
    nonBasic.erase(nonBasic.begin() + p);
    FOR(i, 0, m)
      a[i].erase(a[i].begin() + p);
    c.assign(n, 0);
    FOR(j, 0, n)
    {
      if (nonBasic[j] < n)
        c[j] = _c[nonBasic[j]];
      else
        nonBasic[j]--;
    }
    FOR(i, 0, m)
    {
      if (basic[i] < n)
      {
        v += _c[basic[i]] * b[i];
        FOR(j, 0, n)
          c[j] -= _c[basic[i]] * a[i][j];
      }
      else
        basic[i]--;
    }
  }
  pair<VD, db> simplex(const vector<VD>& _a, const VD& _b, const VD& _c)
  {
    initializeSimplex(_a, _b, _c);
    assert(SZ(a) == m);
    FOR(i, 0, m)
      assert(SZ(a[i]) == n);
    assert(SZ(b) == m);
    assert(SZ(c) == n);
```

```
    assert(SZ(nonBasic) == n);
    assert(SZ(basic) == m);
    findOptimal();
    VD x(n);
    FOR(i, 0, m)
      if (basic[i] < n)
        x[basic[i]] = b[i];
    return {x, v};
  }
private:
  int m, n;
  VI nonBasic, basic;
  vector<VD> a;
  VD b;
  VD c;
  db v;
};
```

Convolutions

conv-xor.hpp

Description: $c_k = \sum_{i \oplus j = k} a_i b_j$.

b80d13, 24 lines

```
void convXor(VI& a, int k)
{
  FOR(i, 0, k)
    FOR(j, 0, 1 << k)
      if((j & (1 << i)) == 0)
      {
        int u = a[j];
        int v = a[j + (1 << i)];
        a[j] = add(u, v);
        a[j + (1 << i)] = sub(u, v);
      }
}
VI multXor(VI a, VI b, int k)
{
  convXor(a, k);
  convXor(b, k);
  FOR(i, 0, 1 << k)
    a[i] = mult(a[i], b[i]);
  convXor(a, k);
  int d = inv(1 << k);
  FOR(i, 0, 1 << k)
    a[i] = mult(a[i], d);
  return a;
}
```

conv-or.hpp

Description: $c_k = \sum_{i \text{ OR } j = k} a_i b_j$.

e4e659, 21 lines

```
void convOr(VI& a, int k, bool inverse)
{
  FOR(i, 0, k)
    FOR(j, 0, 1 << k)
      if((j & (1 << i)) == 0)
      {
        if(inverse)
          updSub(a[j + (1 << i)], a[j]);
        else
          updAdd(a[j + (1 << i)], a[j]);
      }
}
VI multOr(VI a, VI b, int k)
{
  convOr(a, k, false);
  convOr(b, k, false);
  FOR(i, 0, 1 << k)
    a[i] = mult(a[i], b[i]);
  convOr(a, k, true);
  return a;
}
```

subset-convolution.hpp

Description: $c[S] = \sum_{T \subseteq S} a[S] \cdot b[S \setminus T]$.

Time: $\mathcal{O}\left(n^2 \cdot 2^n\right)$, 1.5s for $n = 20$.

5f8849, 27 lines

```
vector<VI> rankedMobius(VI a, int n)
{
  vector<VI> res(n + 1, VI(1 << n));
  FOR(mask, 0, 1 << n)
    res[__builtin_popcount(mask)][mask] = a[mask];
  FOR(sz, 0, n + 1)
    convOr(res[sz], n, false);
  return res;
}
VI subsetConvolution(VI a, VI b, int n)
{
  auto f = rankedMobius(a, n);
  auto g = rankedMobius(b, n);

  vector<VI> conv(n + 1, VI(1 << n));
  FOR(sz, 0, n + 1)
  {
    FOR(i, 0, sz + 1)
      FOR(mask, 0, 1 << n)
        updAdd(conv[sz][mask], mult(f[i][mask], g[sz - i][mask]));
    convOr(conv[sz], n, true);
  }
  VI res(1 << n);
  FOR(mask, 0, 1 << n)
    res[mask] = conv[__builtin_popcount(mask)][mask];
  return res;
}
```

Polynomials and FFT

fft.hpp

Description: $\text{GEN}^{\frac{\text{LEN}}{2}} = \text{mod} - 1$.

ULL $\text{mod} = 9223372036737335297$, $\text{GEN} = 3^{\frac{\text{mod}-1}{\text{LEN}}}$, $\text{LEN} \le 2^{24}$.

d24e3f, 97 lines

```
const int mod = 998244353;

int add(int a, int b)
{
  return a + b < mod ? a + b : a + b - mod;
}
int sub(int a, int b)
{
  return a - b >= 0 ? a - b : a - b + mod;
}
int mult(int a, int b)
{
  return (LL)a * b % mod;
}
int binpow(int a, int n)
{
  int res = 1;
  while(n)
  {
    if(n & 1)
      res = mult(res, a);
    a = mult(a, a);
    n /= 2;
  }
  return res;
}

const int LEN = 1 << 23;
const int GEN = 31;
const int IGEN = binpow(GEN, mod - 2);

//void init()
//{
//  db phi = (db)2 * acos(-1.) / LEN;
//  FOR(i, 0, LEN)
//    pw[i] = com(cos(phi * i), sin(phi * i));
//}

void fft(VI& a, bool inv)
{
  int lg = __builtin_ctz(SZ(a));
  FOR(i, 0, SZ(a))
  {
    int k = 0;
    FOR(j, 0, lg)
      k |= ((i >> j) & 1) << (lg - j - 1);
    if(i < k)
      swap(a[i], a[k]);
  }
  for(int len = 2; len <= SZ(a); len *= 2)
  {
    int ml = binpow(inv ? IGEN : GEN, LEN / len);
    //int diff = inv ? LEN-LEN / len : LEN / len;
    for(int i = 0; i < SZ(a); i += len)
    {
      int pw = 1;
      //int pos = 0;
      FOR(j, 0, len / 2)
      {
        int v = a[i + j];
        int u = mult(a[i + j + len / 2], pw);
        // * pw[pos]

        a[i + j] = add(v, u);
        a[i + j + len / 2] = sub(v, u);

        pw = mult(pw, ml);
        //pos = (pos + diff) % LEN;
      }
    }
  }
  if(inv)
  {
    int m = binpow(SZ(a), mod - 2);
    FOR(i, 0, SZ(a))
```

```
      a[i] = mult(a[i], m);
  }
}

VI mult(VI a, VI b)
{
  int sz = 0;
  int sum = SZ(a) + SZ(b) - 1;
  while((1 << sz) < sum) sz++;
  a.resize(1 << sz);
  b.resize(1 << sz);

  fft(a, false);
  fft(b, false);

  FOR(i, 0, SZ(a))
    a[i] = mult(a[i], b[i]);

  fft(a, true);
  a.resize(sum);
  return a;
}
```

inverse.hpp

Description: $\frac{1}{A(x)}$ modulo x^k.

a4673f, 32 lines

```
VI inverse(const VI& a, int k)
{
  assert(SZ(a) == k && a[0] != 0);
  if(k == 1)
    return {binpow(a[0], mod - 2)};

  VI ra = a;
  FOR(i, 0, SZ(ra))
    if(i & 1)
      ra[i] = sub(0, ra[i]);

  int nk = (k + 1) / 2;
  VI t = mult(a, ra);
  t.resize(k);

  FOR(i, 0, nk)
    t[i] = t[2 * i];

  t.resize(nk);
  t = inverse(t, nk);
  t.resize(k);

  RFOR(i, nk, 1)
  {
    t[2 * i] = t[i];
    t[i] = 0;
  }

  VI res = mult(ra, t);
  res.resize(k);
  return res;
}
```

exp-log.hpp

Description: $\log(A(x))$ and $\exp(A(x))$ modulo x^k.

33cb46, 52 lines

```
VI deriv(const VI& a, int k)
{
  VI res(k);
  FOR(i, 0, k)
    if(i + 1 < SZ(a))
      res[i] = mult(a[i + 1], i + 1);
  return res;
}

VI integr(const VI& a, int k)
{
  VI res(k);
  RFOR(i, k, 1)
    res[i] = mult(a[i - 1], inv[i]);
  res[0] = 0;
  return res;
}

VI log(const VI& a, int k)
{
  assert(a[0] == 1);
  VI ml = mult(deriv(a, k), inverse(a, k));
  return integr(ml, k);
}

VI exp(VI a, int k)
{
  assert(a[0] == 0);

  VI Qk = {1};
  int pw = 1;
  while(pw <= k)
  {
    pw *= 2;

    Qk.resize(pw);
    VI lnQ = log(Qk, pw);

    FOR(i, 0, SZ(lnQ))
    {
      if(i < SZ(a))
        lnQ[i] = sub(a[i], lnQ[i]);
      else
        lnQ[i] = sub(0, lnQ[i]);
    }
    lnQ[0] = add(lnQ[0], 1);

    Qk = mult(Qk, lnQ);
  }
  Qk.resize(k);
  return Qk;
}
```

modulo.hpp

Description: $[\frac{A(x)}{B(x)}]$ and $A(x)$ modulo $B(x)$.

4ccc23, 37 lines

```
void removeLeadingZeros(VI& a)
{
  while(SZ(a) > 0 && a.back() == 0)
    a.pop_back();
}
pair<VI, VI> modulo(VI a, VI b)
{
  removeLeadingZeros(a);
  removeLeadingZeros(b);
  //be careful with this case
  assert(SZ(a) != 0 && SZ(b) != 0);

  int n = SZ(a), m = SZ(b);
  if(m > n)
    return MP(VI{}, a);

  reverse(ALL(a));
  reverse(ALL(b));

  VI d = b;
  d.resize(n - m + 1);
  d = mult(a, inverse(d, n - m + 1));
  d.resize(n - m + 1);

  reverse(ALL(a));
  reverse(ALL(b));
  reverse(ALL(d));

  VI res = mult(b, d);
  res.resize(SZ(a));
  FOR(i, 0, SZ(a))
    res[i] = sub(a[i], res[i]);

  removeLeadingZeros(d);
  removeLeadingZeros(res);
  return MP(d, res);
}
```

multipoint-eval.hpp

Description: build calculates the products of $x - x_i$.
solve calculates the values of $Q(x)$ in $x_0, \ldots, x_{n-1}$.
First call build$(0, 0, n)$, then call solve$(0, 0, n, q)$.

d753bb, 34 lines

```
int x[LEN];
VI p[2 * LEN];

void build(int v, int tl, int tr)
{
  if(tl + 1 == tr)
  {
    p[v] = {sub(0, x[tl]), 1};
    return;
  }
  int tm = (tl + tr) / 2;
  build(2 * v + 1, tl, tm);
  build(2 * v + 2, tm, tr);

  p[v] = mult(p[2 * v + 1], p[2 * v + 2]);
}
int ans[LEN];
void solve(int v, int tl, int tr, const VI& q)
//q != q % p[0] -> wa
{
  if(SZ(q) == 0)
    return;
  if(tl + 1 == tr)
  {
    ans[tl] = q[0];
```

```
    return;
  }
  int tm = (tl + tr) / 2;
  solve(2 * v + 1, tl, tm,
  modulo(q, p[2 * v + 1]).S);

  solve(2 * v + 2, tm, tr,
  modulo(q, p[2 * v + 2]).S);
}
```

Newton's method

Usable to find the solution of equation $F(Q) = 0$.

For example $F(Q) = x \cdot Q^2 + A - Q = 0$.

Newton's method approximates the solution of the equation using the formula:

$$Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)}, where\ F' = \frac{dF}{dQ}$$

Example of the derivative: $F'(Q) = 2 \cdot x \cdot Q - 1$.

Keep in mind that $|Q_k| = 2^k$.

FFT tricks

Two-dimensional FFT

The complexity is $O(nm(\log n + \log m))$. The main problem is to resize the matrix. You must add non-empty vectors.

Divide-and-conquer FFT

Suppose we have the following DP relation:
$f(t) = g(t) - \sum_{0 \le u < t} f(u)h(t-u)$, where $g(t)$ and $h(t)$ are known and we want to compute $f(t)$. We can apply divide-and-conquer FFT.

Let $m = \lfloor \frac{l+r}{2} \rfloor$. We guarantee the following invariant conditions.

By the time we compute the values for the segment $[l, r)$, the following conditions are already met:

- The values for $[0, l)$ on the DP is already determined.
- The sum of contributions from $[0, l)$ through $[l, r)$ is already applied to the DP in $[l, r)$.

When calculate the values for the segment $[l, r)$ do:

- Calculate the values for the segment $[l, m)$ recursively.
- Calculate the contributions from $[l, m)$ to $[m, r)$.
- Calculate the values for the segment $[m, r)$ recursively.

Properties of the discrete Fourier transform

$$DFT(x)_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i 2\pi \frac{k}{N} n}$$

Let $x_n^R = x_{N-n \mod N}$.

$DFT(x^R) = \overline{DFT(x)}$.

For real x, $DFT(x)^R = \overline{DFT(x)}$.

Interpolation

When $x_0, x_1, \ldots, x_d$ and $y_0, y_1, \ldots, y_d$ are given (where x_i are pairwise distinct), a polynomial $f(x)$ of degree no more than d such that $f(x_i) = y_i (i = 0, \ldots, d)$ is uniquely determined.

Lagrange polynomial

Lagrange basis polynomial: $L_i(x) = \prod_{j \ne i} \frac{x - x_j}{x_i - x_j}$.

$f(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_d L_d(x)$.

Newton polynomial

Divided differences:

$[y_i] = y_i$

$[y_i, y_{i+1}] = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$

$[y_i, \ldots, y_j] = \frac{[y_{i+1}, \ldots, y_j] - [y_i, \ldots, y_{j-1}]}{x_j - x_i}$

Newton basis polynomial: $N_i(x) = \prod_{j=0}^{i-1} (x - x_j)$.

$f(x) = [y_0] N_0(x) + \cdots + [y_0, y_1, \ldots, y_d] N_d(x)$.

Linear recurrence

berlekamp-massey.hpp

Description: Finds a sequence of d integers $c_1, \ldots, c_d$ of the minimum length d such that $a_i = \sum_{j=1}^{d} c_j a_{i-j}$.

0f35d6, 36 lines

```
VI berlekampMassey(const VI& a)
{
  VI c = {1}, bp = {1};
  int l = 0, b = 1, x = 1;
  FOR(j, 0, SZ(a))
  {
    assert(SZ(c) == l + 1);
    int d = a[j];
    FOR(i, 1, l + 1)
      updAdd(d, mult(c[i], a[j - i]));
    if (d == 0)
    {
      x++;
      continue;
    }
    VI t = c;
    int coef = mult(d, binpow(b, mod - 2));
    if (SZ(bp) + x > SZ(c))
      c.resize(SZ(bp) + x);
    FOR(i, 0, SZ(bp))
      updSub(c[i + x], mult(coef, bp[i]));
    if (2 * l > j)
    {
      x++;
      continue;
    }
    l = j + 1 - l;
    bp = t;
    b = d;
    x = 1;
  }
  c.erase(c.begin());
  for (int& ci : c)
    ci = mult(ci, mod - 1);
  return c;
}
```

bostan-mori.hpp

Description: computes the n-th term of a given linearly recurrent sequence $a_i = \sum_{j=1}^{d} c_j a_{i-j}$. The problem reduces to determining $[x^n]P(x)/Q(x)$. $\frac{P(x)}{Q(x)} = \frac{P(x)Q(-x)}{Q(x)Q(-x)} = \frac{U_e(x^2)}{V(x^2)} + x \cdot \frac{U_o(x^2)}{V(x^2)}$.

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

966fbd, 41 lines

```
int bostanMori(const VI& c, VI a, LL n) {
  int k = SZ(c);
  assert(SZ(a) == k);
  int m = 1 << (33 - __builtin_clz(k));
  assert(m >= 2 * k + 1);
  VI q(k + 1);
  q[0] = 1;
  FOR(i, 0, k)
    q[i + 1] = sub(0, c[i]);
  VI p = mult(a, q);
  p.resize(m);
  FOR(i, k, m)
    p[i] = 0;
  q.resize(m);
  VI qMinus;
  while (n)
  {
    qMinus = q;
    for (int i = 1; i <= k; i += 2)
      qMinus[i] = sub(0, qMinus[i]);
    fft(qMinus, false);
    fft(p, false);
    fft(q, false);
    FOR(i, 0, m)
      p[i] = mult(p[i], qMinus[i]);
    fft(p, true);
    FOR(i, 0, m)
      q[i] = mult(q[i], qMinus[i]);
    fft(q, true);
    FOR(i, 0, k)
      p[i] = p[2 * i + (n & 1)];
    FOR(i, k, m)
      p[i] = 0;
    FOR(i, 0, k + 1)
      q[i] = q[2 * i];
    FOR(i, k + 1, m)
      q[i] = 0;
    n >>= 1;
  }
  return mult(p[0], binpow(q[0], mod - 2));
}
```

Mathematical analysis and numerical methods

golden-section-search.hpp

4c0990, 27 lines

```
db goldenSectionSearch(db l, db r)
{
  const db c = (-1 + sqrt(5)) / 2;
  const int M = 474;
  db m1 = r - c * (r - l), fm1 = f(m1),
    m2 = l + c * (r - l), fm2 = f(m2);
  FOR(i, 0, M)
  {
    if (fm1 < fm2)
    {
      r = m2;
      m2 = m1;
      fm2 = fm1;
      m1 = r - c * (r - l);
      fm1 = f(m1);
    }
    else
    {
      l = m1;
      m1 = m2;
      fm1 = fm2;
      m2 = l + c * (r - l);
      fm2 = f(m2);
    }
  }
  return (l + r) / 2;
}
```

Taylor series

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + o((x - x_0)^n)$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \qquad \ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n}$$

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} \qquad \sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Runge-Kutta 4th Order

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y), y(0) = y_0, x_{i+1} - x_i = h$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

$$k_1 = f(x_i, y_i) \qquad k_2 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$$

$$k_3 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h) \qquad k_4 = f(x_i + h, y_i + k_3 h)$$

List of integrals

$$\int \frac{\mathrm{d}x}{a^2 + x^2} = \frac{1}{a} \operatorname{arctg} \frac{x}{a} + C$$

$$\int \frac{\mathrm{d}x}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{x + a}{x - a} \right| + C$$

$$\int \frac{\mathrm{d}x}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a} + C$$

$$\int \frac{\mathrm{d}x}{\sqrt{x^2 + a}} = \ln \left| x + \sqrt{x^2 + a} \right| + C$$

$$\int \frac{\mathrm{d}x}{\cos^2 x} = \operatorname{tg} x + C$$

$$\int \frac{\mathrm{d}x}{\sin^2 x} = -\operatorname{ctg} x + C$$

Simpson's rule

n – even number, $h = \frac{b-a}{n}$, $x_i = a + ih$

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{3} \left[f(x_0) + 4 \sum_{i=1}^{\frac{n}{2}} f(x_{2i-1}) + \right.$$
$$\left. +2 \sum_{i=1}^{\frac{n}{2}-1} f(x_{2i}) + f(x_n) \right]$$

Vandermonde matrix

$$V = V(x_0, x_1, \cdots, x_m) = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix}$$

$$V_{i,j} = x_i^j, \qquad \det(V) = \prod_{0 \le i < j \le n} (x_j - x_i).$$

Hadamard matrix

$$H_1 = \begin{bmatrix} 1 \end{bmatrix}, \qquad H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}$$

$\det(H_n) = \pm n^{\frac{n}{2}}$

For a matrix M such that $|M_{ij}| \le 1$, holds $|\det(M)| \le n^{n/2}$.

Number theory

Calculation of $a^b \mod m$

if $b \ge \phi(m)$, then value $a^b \equiv a^{[b \mod \phi(m)] + \phi(m)} \pmod m$.

Generators

A generator exists only for $n = 1, 2, 4, p^k, 2p^k$ for odd primes p and positive integers k.

g is a generator modulo n if any number coprime with n can be represented as $\left[g^i \mod n\right], 0 \le i < \phi(n)$.

To find a generator:

- find $\phi(n)$ and $p_1, ..., p_m$ — the prime factors of $\phi(n)$
- g is generator only if $g^{\frac{\phi(n)}{p_j}} \not\equiv 1 \pmod n$ for each j
- check $g = 2, 3, 4, ..., p - 1$

Wilson's theorem

p is prime if and only if $(p-1)! \equiv (p-1) \pmod p$.

Quadratic residues

q is a quadratic residue modulo p if there exists an integer x such that $x^2 \equiv q \pmod p$. If p is odd prime then there exist $\frac{p+1}{2}$ residues (including 0).

Number theory functions

$$n = p_1^{\alpha_1} \cdot \dots \cdot p_k^{\alpha_k}$$

$$\phi(n) = \prod p_i^{\alpha_i - 1}(p_i - 1) \text{ – the number of coprimes}$$

$$F(n) = \frac{n \cdot \phi(n)}{2} \text{ – the sum of coprimes for } n > 1$$

$$\mu(n) = (-1)^k \text{ if } \max(\alpha_i) = 1 \text{, else } 0$$

$$\sigma_k(n) = \sum_{d|n} d^k$$

$$\sigma_0(n) = \prod(\alpha_i + 1)$$

$$\sigma_{k>0}(n) = \prod \frac{p_i^{(\alpha_i+1)\cdot k} - 1}{p_i^k - 1}$$

Möbius

$$g(n) = \sum_{d|n} f(d) \iff f(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right)$$

$$M(n) = \sum_{k=1}^{n} \mu(k), \qquad \sum_{d=1}^{n} M\left(\left\lfloor \frac{n}{d} \right\rfloor\right) = 1$$

$$\sum_{d|n} \phi(d) = n, \qquad \sum_{d|n} \mu(d) = [n = 1]$$

Combinatorics

Binomials

$$\sum_{k=0}^{n} C_n^k = 2^n \qquad \sum_{k=0}^{m} C_{n+k}^k = C_{n+m+1}^m$$

$$\sum_{m=0}^{n} C_m^k = C_{n+1}^{k+1} \qquad \sum_{k=0}^{n} (C_n^k)^2 = C_{2n}^n$$

$$\sum_{j=0}^{k} C_m^j C_{n-m}^{k-j} = C_n^k \qquad \sum_{j=0}^{m} C_m^j C_{n-m}^{k-j} = C_{n+1}^{k+1}$$

$$\sum_{k=0}^{n} C_{n-k}^k = F_{n+1}$$

Catalan numbers

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1} C_{2n}^n = C_{2n}^n - C_{2n}^{n-1}$$

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786

Fibonacci numbers

$$F_1 = F_2 = 1 \qquad \gcd(F_m, F_n) = F_{\gcd(n,m)}$$

$$F_n = F_{n-1} + F_{n-2} \qquad F_{n+1}F_{n-1} - F_n^2 = (-1)^n$$

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n \qquad F_{47} \approx 2.9 \cdot 10^9$$

$$F_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}} \qquad F_{88} \approx 1.1 \cdot 10^{18}$$

Stirling numbers of the second kind

$S(n,k)$ – the number of ways to divide n element into k non-empty groups.

$S(n,n) = 1$, $n \geq 0$

$S(n,0) = 0$, $n > 0$

$S(n,k) = S(n-1,k-1) + S(n-1,k) \cdot k$.

$B_n = \sum_{k=0}^{n} S(n,k)$ from $n = 0$:

1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, 1382958545, 10480142147, 82864869804,...

Generating functions

$$[x^i](1+x)^n = C_n^i \quad [x^i](1-x)^{-n} = C_{n+i-1}^i$$

$$C_\alpha^n = \frac{\alpha(\alpha-1)\dots(\alpha-n+1)}{n!}$$

$$\prod_{n=1}^{\infty}(1-x^n) = \sum_{k=-\infty}^{\infty} (-1)^k x^{\frac{k(3k-1)}{2}} \text{(pentagonal number theorem)}$$

Hook length formula

A standard Young tableau is a filling of the n cells of the Young diagram with a permutation, such that each row and each column form increasing sequences. The **hook** $h_\lambda(i,j)$ is number of cells (a,b) in diagram such that $a = i$ and $b \geq j$ or $a \geq i$ and $b = j$.

7	4	3	1
5	2	1	
2			
1			

A tableau listing the hook length of each cell in the Young diagram $(4,3,1,1)$

The number of standard Young tableaux of shape λ:

$$f^\lambda = \frac{n!}{\prod h_\lambda(i,j)}$$

Burnside's lemma

Let G be a finite group that acts on a set X.

The *orbit* of an element x in X is the set of elements in X to which x can be moved by the elements of G. The orbit of x is denoted by $G \cdot x$:

$$G \cdot x = \{g \cdot x \mid g \in G\}.$$

For each g in G, let X^g denote the set of elements in X that are fixed by g (also said to be left invariant by g), that is, $X^g = \{x \in X \mid g \cdot x = x\}$. Burnside's lemma asserts the following formula for the number of orbits, denoted $|X/G|$:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

Graphs

Prüfer sequence

At step i, remove the leaf with the smallest label and set the i-th element of the Prüfer sequence to be the label of this leaf's neighbour. The Prüfer sequence of a labeled tree is unique and has length $n-2$.

The number of spanning trees of K_n is n^{n-2}.
The number of spanning trees of $K_{L,R}$ number is $L^{R-1} \cdot R^{L-1}$.

Let $T_{n,k}$ be the number of labelled forests on n vertices with k connected components, such that vertices $1,\dots,k$ all belong to different components. $T_{n,k} = k \cdot n^{n-k-1}$.

The number of spanning trees in a complete graph K_n with the fixed degrees d_i is equal to: $\frac{(n-2)!}{\prod(d_i-1)}$

For a forest graph with connected components of sizes $s_0,\dots,s_{k-1}$, the number of ways to add edges to make a spanning tree is equal to: $n^{k-2} \cdot \prod s_i$

Chromatic polynomial

For a graph G, $\chi(G,\lambda) = \chi(\lambda)$ counts the number of its vertex λ-colorings. There is a unique polynomial $\chi(\lambda)$.
Deletion-contraction:

- The graph G/uv is obtained by merging u and v.
- The graph $G - uv$ is obtained by deleting the edge uv.
- $\chi(G,\lambda) = \chi(G-uv,\lambda) - \chi(G/uv,\lambda)$.

G is tree	$\chi(\lambda) = \lambda(\lambda-1)^{n-1}$
G is cycle C_n	$\chi(\lambda) = (\lambda-1)^n + (-1)^n(\lambda-1)$

Proposition. *$\chi(\lambda)$ is equal to the number of pairs (σ,O), where σ is any map $\sigma: V \to \{1,\dots,\lambda\}$ and O is an orientation of G, subject to the two conditions:*

- *The orientation O is acyclic.*
- *If $u \to v$ in O, then $\sigma(u) > \sigma(v)$.*

Define $\overline{\chi}(\lambda)$ to be the number of pairs (σ,O), where σ is any map $\sigma: V \to \{1,\dots,\lambda\}$ and O is an orientation of G, subject to the two conditions:

- The orientation O is acyclic.
- If $u \to v$ in O, then $\sigma(u) \geq \sigma(v)$.

Theorem. *Suppose that $|V| = n$. Then for all non-negative integers λ holds:*

$$\overline{\chi}(\lambda) = (-1)^n \chi(-\lambda)$$

Corollary. *$(-1)^n \chi(G,-1)$ is equal to the number of acyclic orientations of G.*

Kirchhoff's theorem

Let G be a finite graph, allowing multiple edges but not loops.

The laplacian matrix L of G is the $n \times n$ matrix whose (i,j)-entry L_{ij} is given by

$$L_{ij} = \begin{cases} -m_{ij}, & \text{if } i \neq j,\ m_{ij} \text{ edges between } v_i \text{ and } v_j, \\ \deg(v_i), & \text{if } i = j. \end{cases}$$

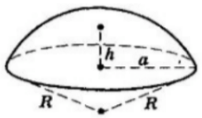
Let L_0 denote L with the i-th row and column removed for any i. Then for a connected graph, $\det(L_0)$ equals the number of spanning trees of G.

Geometry

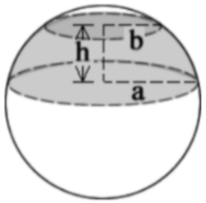
Trigonometry formulas

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

Ball formulas



$$a = \sqrt{h \cdot (2R - h)}$$
$$V = \pi \cdot h^2 (R - \tfrac{h}{3})$$



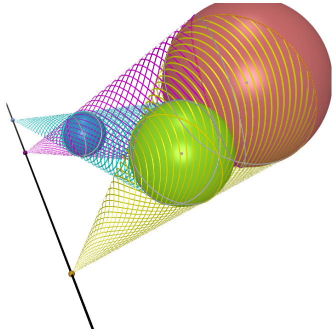
$$V = \tfrac{1}{6}\pi h(3a^2 + 3b^2 + h^2)$$
$$R = \sqrt{\tfrac{((a-b)^2+h^2)((a+b)^2+h^2)}{4h^2}}$$

Triangle formulas

$$S = \sqrt{p(p-a)(p-b)(p-c)} = \frac{abc}{4R}$$
$$m_a^2 = \frac{2b^2 + 2c^2 - a^2}{4}\text{(median)}$$
$$w_a^2 = \frac{bc((b+c)^2 - a^2)}{(b+c)^2}\text{(bisector)}$$
$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$
$$a^2 = b^2 + c^2 - 2bc \cos A$$

Monge's theorem

There are three circles(balls) of different radii, for each pair of circles find the point of intersection of the external tangents. All three obtained points **lie on a line**. The point from the pair of the largest and the smallest **lies between** the other two.



Pick's theorem

Suppose that a polygon has integer coordinates for all of its vertices. Let i be the number of integer points inside, and let b be the number of integer points on boundary. Then the area $S = i + \frac{b}{2} - 1$.

Ptolemy's theorem

For a general quadrilateral $ABCD$ holds:
$AB \cdot CD + AD \cdot BC \geq AC \cdot BD$.

Equality holds if and only if the quadrilateral is cyclic.

Ceva's theorem

Given a triangle $\triangle ABC$ with a point P inside the triangle, continue lines AP, BP, CP to hit BC, CA, AB at D, E, F, respectively. Ceva's theorem states that $\frac{AF}{FB} \cdot \frac{BD}{DC} \cdot \frac{CE}{EA} = 1$.

Simson line

Given a triangle $\triangle ABC$ and a point P on its circumcircle, the three closest points to P on lines AB, AC, and BC are collinear. The line through these points is the Simson line of P.

Euler line

For a general triangle, the orthocenter H, the centroid G, and the circumcenter O, in this order, lie on the same line (Euler line) and $\frac{|HG|}{|GO|} = \frac{2}{1}$.

Platonic solids

Polyhedron	Vertices	Edges	Faces
tetrahedron	4	6	4
cube	8	12	6
octahedron	6	12	8
dodecahedron	20	30	12
icosahedron	12	30	20

Gaussian integer

gaussian-integer.hpp

Description: $n = am + b$, $\frac{n}{m} = a, n\%m = b$. use __gcd instead of gcd.
Facts: Primes of the form $4n+3$ are Gaussian primes. Uniqueness of prime factorization.

cb938e, 41 lines

```
LL closest(LL u, LL d)
{
  if(d < 0)
    return closest(-u, -d);
  if(u < 0)
    return -closest(-u, d);
  return (2 * u + d) / (2 * d);
}
struct num : complex<LL>
{
  num(LL a, LL b = 0) : complex(a, b) {}
  num(complex a) : complex(a) {}
  num operator/ (num x)
  {
    num prod = *this * conj(x);
    LL D = (x * conj(x)).real();

    LL m = closest(prod.real(), D);
    LL n = closest(prod.imag(), D);

    return num(m, n);
  }
  num operator% (num x)
  {
    return *this - x * (*this / x);
  }
  bool operator == (num b)
  {
    FOR(it, 0, 4)
    {
      if(real() == b.real() && imag() == b.imag())
        return true;
      b = b * num(0, 1);
    }
    return false;
  }
  bool operator != (num b)
  {
    return !(*this == b);
  }
};
```

Nim product

nim-product.hpp

Description: The Nim sum $\oplus$: $a \oplus b := \text{mex}(\{a' \oplus b | a' < a\} \cup \{a \oplus b' | b' < b\})$. The Nim product $\otimes$: $a \otimes b := \text{mex}\{(a' \otimes b) \oplus (a \otimes b') \oplus (a' \otimes b') | a' < a, b' < b\}$. Let A be the set consisting of integers between 0 (inclusive) and 2^{2^n} (exclusive) (where n is an integer). Then the algebraic structure whose addition is $\oplus$ and multiplication is $\otimes$ forms a field. Such a field is called **Nimber**.

9d1a59, 30 lines

```
typedef unsigned long long ULL;

const int S = 8;

int small[1 << S][1 << S];

void init()
{
  FOR(i, 0, 1 << S)
    FOR(j, 0, 1 << S)
      small[i][j] = -1;
}

ULL nimProduct(ULL a, ULL b, int p = 64)
{
```

```
  if (min(a, b) <= 1)
    return a * b;
  if (p <= S && small[a][b] != -1)
    return small[a][b];
  p >>= 1;
  ULL a1 = a >> p, a2 = a & ((1ULL << p) - 1);
  ULL b1 = b >> p, b2 = b & ((1ULL << p) - 1);
  ULL c = nimProduct(a1, b1, p);
  ULL d = nimProduct(a2, b2, p);
  ULL e = nimProduct(a1 ^ a2, b1 ^ b2, p);
  ULL res = nimProduct(c, 1ULL << (p - 1), p) ^ d ^ ((d ^ e) << p);
  if (p <= S / 2)
    small[a][b] = res;
  return res;
}
```