Ivan Franko National University of Lviv

Stallions

Maksym Shcherba, Petro Tarnavskyi, Yarema Stiahar

2023-11-03

Contents

Contest (1)

template.hpp

27 lines

```
//hash = 7d0184

#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);
  cout << fixed << setprecision(15);

  return 0;
}
```

compilation.txt

2 lines

```
g++ -O2 -std=c++17 -Wno-unused-result -Wshadow -
    Wall -o %e %e.cpp
g++ -std=c++17 -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
```

1.1 Rules

Don't code solution without proof.

Try to find counter-tests.

Discuss realisation, try to assist.

Freeze time: Discuss how much problem we need/want to solve. At beginning (and after AC) discuss situation and what to do.

1.2 Troubleshoot

Pre-submit

F9. Write a few manual test cases. Calculate time and memory complexity. Check limits. Check overflows, size of arrays, 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 collitions. Have you understood the problem correctly? Write brute and generator.

Runtime error

F9. Print your solution! Read your code. F9 with generator.

Time limit exceeded

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

Memory limit exceeded

Calculate memory usage with stack in recurtion.

Data Structures (2)

dsu.hpp

2de4ff, 34 lines

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

  void init(int _n)
  {
    n = _n;
    sz.assign(n, 1);
    p.resize(n);
    iota(ALL(p), 0);
  }

  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

d4ebda, 43 lines

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

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

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

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

  int lower_bound(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 + v[j] < x)
      {
        sum += v[j];
        i = j;
      }
      lg--;
    }
    return i + 1;
  }
};
```

Fenwick.txt

20 lines

```
Minimum on segment:
1) Use two Fenwick trees with n = 2^k.
You can use if n > 1:
n = 1 << (32 - __builtin_clz(n - 1));
2) One tree for normal array and one for reversed
3) When querying for minimum on the segment
only consider segments [(i & (i + 1)), i]
from trees that are COMPLETELY inside [l, r]

Fenwick tree for adding on segment (prefixes):
1) Use 2 arrays: mult and add
2) upd(int i, int updMult, int updAdd)
default Fenwick update.
3) add x on segment [l, r]:
  upd(l, x, -x * (l - 1));
  upd(r, -x, x * r);
4) to calculate sum on prefix r:
  sumAdd and sumMult - default Fenwick sum
  st - initial value of r
  ans = st * sumMult + sumAdd
```

treap.hpp

Description: uncomment in split for explicit key or in merge for implicit priority.

925cdb, 145 lines

```
mt19937 rng;

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

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

struct Treap
{
  Node A[N];
  int sz = 0;

  int getCnt(int v)
  {
    if (v == -1)
      return 0;
    return A[v].cnt;
  }
  int getMn(int v)
  {
    if (v == -1)
      return INF;
    return A[v].mn;
  }
```

```
  int newNode(int val)
  {
    A[sz].init(val);
    return sz++;
  }
  void upd(int v)
  {
    if (v == -1)
      return;
    A[v].cnt = getCnt(A[v].l) +
    getCnt(A[v].r) + 1;

    A[v].mn = min(A[v].x,
    min(getMn(A[v].l), getMn(A[v].r)));
  }
  void reverse(int v)
  {
    if (v == -1)
      return;
    A[v].rev ^= 1;
  }
  void push(int v)
  {
    if (v == -1 || A[v].rev == 0)
      return;
    reverse(A[v].l);
    reverse(A[v].r);
    swap(A[v].l, A[v].r);
    A[v].rev = 0;
  }
  PII split(int v, int cnt)
  {
    if (v == -1)
      return {-1, -1};
    push(v);
    int left = getCnt(A[v].l);
    PII res;
    // if (val <= A[v].x)
    if (cnt <= left)
    {
      if (A[v].l != -1)
        A[A[v].l].par = -1;
      res = split(A[v].l, cnt);
      A[v].l = res.second;
      if (res.second != -1)
        A[res.second].par = v;
      res.second = v;
    }
    else
    {
      if (A[v].r != -1)
        A[A[v].r].par = -1;
      // split(v, val)
      res = split(A[v].r, cnt - left - 1);
      A[v].r = res.first;
      if (res.first != -1)
        A[res.first].par = v;
      res.first = v;
    }
    upd(v);
    return res;
  }
  int merge(int v, int u)
  {
    if (v == -1) return u;
    if (u == -1) return v;
    int res;
    // if (rng() % (getCnt(v) + getCnt(u)) <
      getCnt(v))
    if (A[v].y > A[u].y)
    {
      push(v);
      if (A[v].r != -1)
        A[A[v].r].par = -1;
      res = merge(A[v].r, u);
      A[v].r = res;
      if (res != -1)
        A[res].par = v;
      res = v;
    }
    else
    {
      push(u);
      if (A[u].l != -1)
        A[A[u].l].par = -1;
      res = merge(v, A[u].l);
      A[u].l = res;
      if (res != -1)
        A[res].par = u;
      res = u;
    }
    upd(res);
    return res;
  }
  int getIdx(int v, int from = -1)
  {
    if (v == -1)
      return 0;
    int x = getIdx(A[v].par, v);
    if (from == -1 || A[v].r == from)
      x += getCnt(A[v].l) + 1;
    push(v);
    return x;
  }
};
```

ordered-set.hpp

8 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.
```

sparse-table.hpp

7fdd30, 30 lines

```
int lg[N + 1];

struct SparseTable
{
  int t[N][LOG];

  void init(const VI& v)
  {
    lg[1] = 0;
    FOR (i, 2, N + 1) lg[i] = lg[i / 2] + 1;
    FOR (i, 0, N) FOR (j, 0, LOG) t[i][j] = INF;
    FOR (i, 0, SZ(v)) t[i][0] = v[i];

    FOR (j, 1, LOG)
    {
      int len = 1 << (j - 1);
      FOR (i, 0, N - (1 << j))
      {
        t[i][j] = min(t[i][j - 1],
        t[i + len][j - 1]);
      }
    }
  }

  int query(int l, int r)
  {
    int i = lg[r - l + 1];
```

```
    return min(t[l][i], t[r - (1 << i) + 1][i]);
  }
} st;
```

convex-hull-trick.hpp

06db0c, 74 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 lIt = *itP;
      lIt.xLast = itP->intersect(l);
      erase(itP);
      insert(lIt);
    }
    l.xLast = it == end() ? LINF :
      l.intersect(*it);
    insert(l);
  }
  LL getMaxY(LL x)
  {
    return lower_bound(x)->getY(x);
  }
};
```

Graphs (3)

3.1 Decompositions

centroid.hpp

5dd72c, 46 lines

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

void build(int cent)
{
  dfsSZ(cent, -1);
  int szAll = sz[cent];
  int pr = cent;
  while (true)
  {
    int v = -1;
    for (auto to : g[cent])
    {
      if (to == pr || usedc[to])
        continue;
      if (sz[to] * 2 > szAll)
      {
        v = to;
        break;
      }
    }
    if (v == -1)
      break;
    pr = cent;
    cent = v;
  }
  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 HLD. Vertex v has index tin[v]. To update on path use process as in get().

0031c1, 66 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 x, int y)
{
  LL res = 0;
  while(true)
  {
    int tx = top[x];
    int ty = top[y];
    if (tx == ty)
    {
      int t1 = tin[x];
      int t2 = tin[y];
      if (t1 > t2)
        swap(t1, t2);
      res += query(t1, t2);
      break;
    }
    if (h[tx] < h[ty])
    {
      swap(tx, ty);
      swap(x, y);
    }
    res += query(tin[tx], tin[x]);
    x = p[tx];
  }
  return res;
}
```

biconnected-components.hpp

2d79e1, 83 lines

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

  VI tin, low;
  VI col;
  VI par;
  VI used;
  int t = 1, c = 1;
  vector<int> st;

  int n, m;

  void init(int _n, int _m)
  {
    n = _n;
    m = _m;

    edges.assign(m, {0, 0});
    g.assign(n, {});
    tin.assign(n, 0);
    used.assign(n, 0);
    par.assign(n, -1);
    used.assign(n, 0);

    t = c = 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 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]))
        {
          while (st.back() != e)
          {
            col[st.back()] = c;
            st.pop_back();
          }
          col[st.back()] = c++;
          st.pop_back();
        }
      }
      else
      {
        low[v] = min(low[v], tin[to]);
        if (tin[to] < tin[v])
          st.PB(e);
      }
    }
  }
};
```

dominator-tree.hpp

Description: works for cyclic graphs. Add direct edges to g and reversed edges to gr. dom - immidiate dominator. sdom - semidominator. dom[root] = -1. dom[v] = -1 if v is unreachable.
Time: $\mathcal{O}(n)$

415a75, 95 lines

```
VI g[N];
VI gr[N];
int par[N];  // parent in dfs
bool used[N];
int p[N];    // parent in dsu
int val[N];    // vertex with min sdom in dsu
```

```
int sdom[N]; // min vertex with alternate path
int dom[N];  // immediate dominator
VI bkt[N]; // vertices with this sdom
int tin[N];
int T;
int n;
VI ord;

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 val[v]; // return vertex with min sdom
}

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])
    {
      if (!used[from])
        continue; // don't consider unreachable
            vertices
      if (tin[sdom[v]] > tin[sdom[get(from)]]) //
          find min sdom
      {
        sdom[v] = sdom[get(from)];
      }
    }
    if (v != s)
      bkt[sdom[v]].PB(v);
    for (auto y : bkt[v])
    {
      int u = get(y);
      if (sdom[y] == sdom[u])
        dom[y] = sdom[y]; // if sdoms equals then
             this is dom
      else dom[y] = u; // else we will find it
          later
    }

    if (par[v] != -1)
      p[v] = par[v]; // add vertex to dsu
  }

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

3.2 Flows

kuhn.hpp

Time: 0.6s for $|V| = 10^5, |E| = 2 * 10^5$

39cb20, 81 lines

```
struct Graph
{
  int L, R;
  //edges from left to right in 0 indexing
  vector<VI> g;
  VI mt, P, U;

  void init(int l, int r)
  {
    L = l, R = r;
    g.resize(L);
    P.resize(L);
    U.resize(L);

    mt.resize(R);
  }

  void addEdge(int from, int to)
  {
    assert(0 <= from && from < L);
    assert(0 <= to && to < R);

    g[from].PB(to);
  }

  int iter;
  bool kuhn(int v)
  {
    if (U[v] == iter) return false;
    U[v] = iter;
    random_shuffle(ALL(g[v]));
    for(int to : g[v])
    {
      if (mt[to] == -1)
      {
        mt[to] = v;
        P[v] = to;
        return true;
      }
    }
    for(int to : g[v])
    {
      if (kuhn(mt[to]))
      {
        mt[to] = v;
        P[v] = to;
        return true;
      }
    }
    return false;
  }
  int doKuhn()
  {
```

```
    fill(ALL(mt), -1);
    fill(ALL(P), -1);
    fill(ALL(U), -1);

    int res = 0;
    iter = 0;
    VI order(L);
    iota(ALL(order), 0);
    random_shuffle(ALL(order));

    while(true)
    {
      iter++;
      bool ok = false;
      for(int v : order)
      {
        if (P[v] == -1)
          if (kuhn(v))
          {
            ok = true;
            res++;
          }
      }
      if (!ok) break;
    }
    return res;
  }
};
```

dinic.hpp

6afa18, 93 lines

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

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

  Graph() : _n(0) {}
  Graph(int n) : _n(n), g(n), d(n), p(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;
  }
};
```

min-cost-flow.hpp

8a8605, 103 lines

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

  int _n;
  vector<Edge> edges;
  vector<VI> g;
  vector<LL> d;
  VI p, w;

  Graph(): _n(0) {}
  Graph(int n): _n(n), g(n), d(n), p(n), w(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(p), -1);
      fill(ALL(w), 0);
      queue<int> q1, q2;
      w[s] = 1;
      d[s] = 0;
      q2.push(s);
      while (!q1.empty() || !q2.empty())
      {
        int v;
        if (!q1.empty())
        {
          v = q1.front();
          q1.pop();
        }
        else
        {
          v = q2.front();
          q2.pop();
        }
        for (int e : g[v])
        {
          if (edges[e].flow == edges[e].cap)
            continue;
          int to = edges[e].to;
          LL newDist = d[v] + edges[e].cost;
          if (newDist < d[to])
          {
            d[to] = newDist;
            p[to] = e;
            if (w[to] == 0)
              q2.push(to);
            else if (w[to] == 2)
              q1.push(to);
            w[to] = 1;
          }
        }
        w[v] = 2;
      }
      if (p[t] == -1)
        break;
      int curFlow = INF;
      LL curCost = 0;
      for (int v = t; v != s;)
      {
        int e = p[v];
        curFlow = min(curFlow,
        edges[e].cap - edges[e].flow);
        curCost += edges[e].cost;
        v = edges[e].from;
      }
      for (int v = t; v != s;)
      {
        int e = p[v];
        edges[e].flow += curFlow;
        edges[e ^ 1].flow -= curFlow;
        v = edges[e].from;
      }
      flow += curFlow;
      cost += curCost * curFlow;
    }
    return {flow, cost};
  }
};
```

3.3 Flows text

3.3.1 Recover

Min cut

To restore min cut use search from S on edges with flow $\neq$ capacitie. Original edges from used vertices to unused is minimal cut.

Min vertex cover

Only in bipartite graphs. Minimum number of vertex to cover **edges** equal to size of mathing.

To restore min vertex cover make directed graph:

- mathced edges direct from R to L
- unmathced edges direct from L to R

From unmathced vertices from left do traversal. Cover have vertices from matching:

- unvisited vertices in L
- visited vertices in R

Max independent set

Only in bipartite graphs.

Maximal independent set is complement of min vertex cover

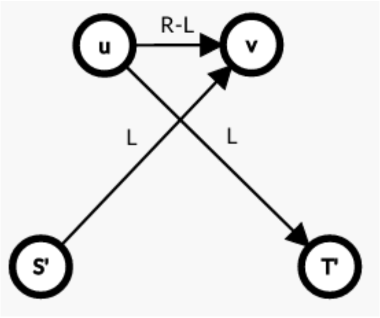
Flow with lower bound

`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.



Binary optimization

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

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.

3.4 Specific

hungarian.hpp

0baccf, 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);
    vector<int> 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;
}
```

edmonds-blossom.hpp

76c9ac, 127 lines

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

  Graph() {}
  Graph(int _n): n(_n), g(_n + 1), label(_n + 1),
    first(_n + 1), mate(_n + 1) {}

  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)
      {
        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)
          {
            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;
  }
};
```

Strings (4)

aho-corasick.hpp

a46c9f, 72 lines

```
const int AL = 26;

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

  void init()
  {
    c = -1;
    p = -1;
    fill(g, g + AL, -1);
    fill(nxt, nxt + AL, -1);
    link = -1;
  }
};

struct AC
{
  vector<Node> a;
  int sz;
  void init(int n)
  {
    a.resize(n);
    a[0].init();
    sz = 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[sz].init();
        a[sz].c = c;
        a[sz].p = v;
        sz++;
      }
      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

0264b8, 66 lines

```
const int AL = 26;

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

struct Automaton
{
  vector<Node> a;
  int sz;
  int head;
  void init(int n)
  {
    a.resize(2 * n);
    a[0].init();
    sz = 1;
    head = 0;
  }
  void add(char c)
  {
    // change to [0 AL)
    int ch = c - 'a';
    int nhead = sz++;
    a[nhead].init();
    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[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

d69c2d, 60 lines

```
void countSort(VI& p, const VI& c)
{
  int n = SZ(p);
  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(const string& t)
{
  string s = t + "$";
  int n = SZ(s);
  VI p(n), c(n);
  FOR (i, 0, n) p[i] = i;
  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());
  return p;
}
```

lcp.hpp

72ff1e, 24 lines

```
VI lcpArray(const string& s, const VI& sa)
{
    int n = SZ(s);
    VI rnk(n);
    FOR (i, 0, n)
        rnk[sa[i]] = i;
    VI lcp(n - 1);
    int h = 0;
    FOR (i, 0, n)
    {
        if (h > 0)
      h--;
        if (rnk[i] == 0)
      continue;
        int j = sa[rnk[i] - 1];
        for (; j + h < n && i + h < n; h++)
        {
            if (s[j + h] != s[i + h])
        break;
        }
        lcp[rnk[i] - 1] = h;
    }
    return lcp;
}
```

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: d1[i] - half-length of odd length palindrome with center in i. d2[i] - half-length of even length palindrome if i is right center of it.

2f1541, 39 lines

```
int d1[N], d2[N];

void manacher(const string& s)
{
    int n = SZ(s);
    int l = -1;
    int r = -1;
    FOR (i, 0, n)
    {
        if (i <= r)
            d1[i] = min(r - i + 1,
              d1[l + (r - i)]);
        while (i + d1[i] < n && i - d1[i] >= 0
          && s[i + d1[i]] == s[i - d1[i]])
            d1[i]++;
        if (i + d1[i] - 1 > r)
        {
            r = i + d1[i] - 1;
```

```
            l = i - (d1[i] - 1);
        }
    }
    l = -1;
    r = -1;
    FOR (i, 0, n)
    {
        if (i <= r)
            d2[i] = min(r - i + 1,
              d2[l + (r - i) + 1]);
        while (i + d2[i] < n
          && i - (d2[i] + 1) >= 0
          && s[i + d2[i]] == s[i - (d2[i] + 1)])
            d2[i]++;
        if (i + d2[i] > r)
        {
            r = i + d2[i] - 1;
            l = i - d2[i];
        }
    }
}
```

palindrome-tree.hpp

c4e179, 64 lines

```
struct Node
{
  int to[AL];
  int link;
  int len;
  void clear()
  {
    fill(to, to + AL, -1);
    link = -1;
    len = -1;
  }
};
struct PalTree
{
  string s;
  vector<Node> A;
  int sz;
  int last;

  void init(string t)
  {
    A.resize(2 * SZ(t));
    A[0].clear();
    A[1].clear();
    A[1].len = 0;
    A[1].link = 0;
    sz = 2;
    last = 1;
    s = t;
  }
  void add(int idx)
  {
    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;
    }
    assert(cur != -1);
    if (A[cur].to[s[idx] - 'a'] == -1)
    {
      A[cur].to[s[idx] - 'a'] = sz;
      A[sz].clear();
      A[sz].len = A[cur].len + 2;
      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[s[idx] - 'a'];
      A[sz].link = link;
      sz++;
    }
    last = A[cur].to[s[idx] - 'a'];
  }
} pt;
```

Geometry (5)

In general, try to build programs that are resistant to the oddities of floating-point numbers. Imagine that some evil demon is slightly modifying every result you compute in the way that is most likely to make your program fail. And try to write clean code that is clearly correct at first glance. If you need long explanations to justify why your program will not fail, then it is more likely that your program will in fact fail.

Victor Lecomte, Handbook of geometry for competitive programmers

geometry.hpp

c98f0c, 591 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};
  }
};
// Returns the squared absolute value
db sq(const Pt& p)
{
  return p.x * p.x + p.y * p.y;
}
// Returns the absolute value
db abs(const Pt& p)
{
  return sqrt(sq(p));
}
// Returns -1 for negative numbers, 0 for zero,
// and 1 for positive numbers
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
Pt perp(const Pt& p)
{
  return {-p.y, p.x};
}
// Returns the dot product of 'p' and 'q'
db dot(const Pt& p, const Pt& q)
{
  return p.x * q.x + p.y * q.y;
}
// Returns the angle between 'p' and 'q'
db angle(const Pt& p, const Pt& q)
{
  return acos(clamp(dot(p, q) / abs(p) /
    abs(q), (db)-1.0, (db)1.0));
}
// Returns the cross product of 'p' and 'q'
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 a polygon 'v' is convex
bool isConvex(const vector<Pt>& v)
{
  bool hasPos = false, hasNeg = false;
  int n = SZ(v);
  FOR(i, 0, n)
  {
    int o = sgn(orient(v[i], v[(i + 1) % n],
      v[(i + 2) % n]));
    hasPos |= o > 0;
    hasNeg |= o < 0;
  }
  return !(hasPos && hasNeg);
}
// 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);
}
// Polar sort of vectors in 'v' around 'o'
void polarSortAround(const Pt& o, vector<Pt>& v)
{
  sort(ALL(v), [o](const Pt& p, const Pt& q)
  {
    bool hp = half(p - o), hq = half(q - o);
    if (hp != hq)
      return hp < hq;
    int s = sgn(cross(p, q));
    if (s != 0)
      return s == 1;
    return sq(p - o) < sq(q - o);
  });
}
// Returns the distance of the closest points
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;
}
// Example:
```

```cpp
// cout << a + b << " " << a - b << "\n";
ostream& operator<<(ostream& os, const Pt& p)
{
  return os << "(" << p.x << "," << p.y << ")";
}
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) {}
  // The line containing two points ‘p‘ and ‘q‘
  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;
  }
  // Returns the distance from ‘p‘
  db dist(const Pt& p) const
  {
    return abs(side(p)) / abs(n);
  }
  // Returns the squared distance from ‘p‘
  db sqDist(const Pt& p) const
  {
    return side(p) * side(p) / (db)sq(n);
  }
  // Returns the perpendicular line through ‘p‘
  Line perpThrough(const Pt& p) const
  {
    return {p, p + n};
  }
  // Compares ‘p‘ and ‘q‘ by their projection
  bool cmpProj(const Pt& p, const Pt& q) const
  {
    return sgn(cross(p, n) - cross(q, n)) < 0;
  }
  // Returns the orthogonal projection of ‘p‘
  Pt proj(const Pt& p) const
  {
    return p - n * side(p) / sq(n);
  }
  // Returns the reflection of ‘p‘ by the line
  Pt refl(const Pt& p) const
  {
    return p - n * 2 * side(p) / sq(n);
  }
};
// Checks if ‘l1‘ and ‘l2‘ are parallel
bool parallel(const Line& l1, const Line& l2)
{
  return sgn(cross(l1.n, l2.n)) == 0;
}
// Returns the intersection point
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;
}
// Checks if ‘p‘ is in the disk 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)});
}
// Returns the area of triangle abc
db areaTriangle(const Pt& a, const Pt& b,
  const Pt& c)
{
  return abs(cross(b - a, c - a)) / 2.0;
}
// Returns the area of polygon ‘v‘
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
bool inConvexPolygon(const vector<Pt>& v,
  const Pt& 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;
}
// Returns true if ‘p‘ is at least as high as ‘a‘
bool above(const Pt& a, const Pt& p)
{
  return sgn(p.y - a.y) >= 0;
}
// Checks if [pq] crosses the ray from ‘a‘
bool crossesRay(const Pt& a, const Pt& p,
  const Pt& q)
{
  return sgn((above(a, q) - above(a, p))
    * orient(a, p, q)) == 1;
}
```

```cpp
// 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;
}
// Returns the counter-clockwise convex hull
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(), upper.begin() + 1,
    prev(upper.end()));
  return lower;
}
// Returns the indices of tangent points
PII tangetsToConvexPolygon(const vector<Pt>& v,
  const Pt& p)
{
  int n = SZ(v), i = 0;
  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;
}
// Returns the Minkowski sum of two convex
// polygons
vector<Pt> minkowskiSum(const vector<Pt>& v1,
  const vector<Pt>& v2)
{
  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;
}
// 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
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)
```

```cpp
      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;
}
// Returns the circumcenter of triangle abc
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)
  {
    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;
}
// Returns the smallest enclosing circle of 'v'
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};
    }
  }
}
```

Math (6)

gcd.hpp

e001bc, 16 lines

```
int gcd(int a, int b, int& x, int& y)
{
  x = 1, y = 0;
  int x2 = 0, y2 = 1;
  while (b)
  {
    int k = a / b;
    x -= k * x2;
    y -= k * y2;
    a %= b;
    swap(a, b);
    swap(x, x2);
    swap(y, y2);
  }
  return a;
}
```

fast-chinese.hpp

Description: $x\%p_i = m_i, lcm(p) < 10^{18}, p < 10^9$.
no solution -> return -1
Time: $\mathcal{O}(nlog)$

9b392a, 25 lines

```
LL FastChinese(VI m, VI p)
{
  assert(SZ(m) == SZ(p));
  LL aa = p[0];
  LL bb = m[0];
  FOR(i, 1, SZ(m))
  {
    int b = (m[i] - bb % p[i] + p[i]) % p[i];
    int a = aa % p[i];
    int c = p[i];

    int x, y;
    int d = gcd(a, c, x, y);
    if(b % d != 0)
      return -1;
    a /= d;
    b /= d;
    c /= d;
    b = b * (LL)x % c;

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

chinese.hpp

Description: $result < 10^{18}, p < 10^9$.
Time: $\mathcal{O}(n^2)$

c692b5, 36 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];
    int x, y;
    int d = gcd(a, c, x, y);

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

    b = b * x % c;
    m[i] = b;
    p[i] = c;
  }
  LL res = m[n - 1];
  RFOR(i, n - 1, 0)
  {
    res *= p[i];
    res += m[i];
  }
  return res;
}
```

gauss.hpp

Description: a[i].back() is right side element
Time: $\mathcal{O}(m^2 * n)$

e404cd, 50 lines

```
VI Gauss(vector<VI> a)
{
  int n = SZ(a);
  if(n == 0)
    return {};
  int m = SZ(a[0]) - 1;//number of variables
  assert(n >= m);

  int vars = m;
  FOR(i, 0, m)
  {
    if (a[i][i] == 0)
    {
      //for double find row
      //with max abs value
      int row = -1;
      FOR(k, i + 1, n)
      {
        if (a[k][i] != 0)
          row = k;
      }
      if(row == -1)
      {
        //variable i can be any
        vars--;
        continue;
      }
      swap(a[i], a[row]);
    }
    int d = inv(a[i][i]);
    FOR(k, i + 1, n)
    {
      int c = mult(a[k][i], d);
      FOR(j, 0, m + 1)
        updSub(a[k][j], mult(c, a[i][j]));
    }
  }
  FOR(i, vars, n)
    if(a[i].back() != 0)
      cout << "No solution\n";
  VI x(m);
  RFOR(i, m, 0)
  {
    x[i] = a[i].back();
    FOR(j, i + 1, m)
      updSub(x[i], mult(a[i][j], x[j]));
    x[i] = mult(x[i], inv(a[i][i]));
  }
  return x;
}
```

miller-rabin.hpp

Description: to speed up change candidates to at least 4 random values rng()
use __int128 in mult

62f1a5, 33 lines

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

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

pollard.hpp

Description: uses Miller-Rabin test. rho finds divisor of n. use __int128 in mult. works in O($n^{1/4}$ * log n).

28f253, 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 * sqrt(sqrt(n));
  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;
}
```

simplex.hpp

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

3805fb, 142 lines

```
struct Simplex
{
private:
  int m, n;
  VI nonBasic, basic;
  vector<vector<db>> a;
  vector<db> b;
  vector<db> c;
  db v;

public:
  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()
  {
    vector<db> 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<vector<db
    >>& _a, const vector<db>& _b, const vector<
    db>& _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<vector<db>, db> simplex(const vector<
    vector<db>>& _a, const vector<db>& _b,
    const vector<db>& _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();
  vector<db> x(n);
  FOR(i, 0, m)
    if (basic[i] < n)
      x[basic[i]] = b[i];
    return {x, v};
  }
};
```

Convolutions (7)

fft.hpp

44d94c, 87 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 a * (LL) 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 fft(VI& a, bool inv)
{
  int lg = 0;
  while((1 << lg) < SZ(a)) lg++;
  FOR(i, 0, SZ(a))
  {
    int x = 0;
    FOR(j, 0, lg)
      x |= ((i >> j) & 1) << (lg - j - 1);
    if(i < x)
      swap(a[i], a[x]);
```

```
  }
  for(int len = 2; len <= SZ(a); len *= 2)
  {
    int ml = binpow(inv ? IGEN : GEN, LEN / len);
    for(int i = 0; i < SZ(a); i += len)
    {
      int pw = 1;
      FOR(j, 0, len / 2)
      {
        int v = a[i + j];
        int u = mult(a[i + j + len / 2], pw);

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

        pw = mult(pw, ml);
      }
    }
  }
  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, 0);
  fft(b, 0);

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

  fft(a, 1);
  a.resize(sum);
  return a;
}
```

inverse.hpp

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

5549eb, 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]);
    }
    updAdd(lnQ[0], 1);

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

modulo.hpp

8b6a95, 34 lines

```
void removeLeadingZeros(VI& a)
{
  while(SZ(a) > 0 && a.back() == 0)
    a.pop_back();
}

pair<VI, VI> modulo(VI a, VI b)
{
  //assert(a.back() != 0 && b.back() != 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

8f6f41, 33 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.hpp

9ffaac, 50 lines

```
VI newton(VI a, int k)
{
  //c_n= a_n + sum(i = 0, n - 1) c_i * c_(n-1-i)
  //Q = A + x * Q * Q
  //F(Q) = Q - x * Q * Q - A
  //F'(Q) = 1 - 2 * x * Q

  VI Qk = {a[0]};
  int pw = 1;
  while(pw <= k)
  {
    assert(SZ(Qk) == pw);
    pw *= 2;

    VI F1(pw);
    F1[0] = 1;
    FOR(i, 0, pw / 2)
      F1[i + 1] = sub(0, mult(2, Qk[i]));
    //F' = 1 - 2 * x * Q

    VI F = mult(Qk, Qk);
    F.resize(pw);
    RFOR(i, pw, 1)
      F[i] = sub(0, F[i - 1]);
    F[0] = 0; // F = -x * Q*Q

    FOR(i, 0, pw / 2)
      F[i] = add(F[i], Qk[i]);
      //F = Q - x * Q * Q
    FOR(i, 0, min(pw, SZ(a)))
      F[i] = sub(F[i], a[i]);
      //F = Q - x * Q * Q - A

    F = mult(F, inverse(F1, pw));
    F.resize(pw);

    FOR(i, 0, pw)
      F[i] = sub(0, F[i]);//-F/F'
    FOR(i, 0, pw / 2)
      F[i] = add(F[i], Qk[i]);//Q - F/F'

    //new Qk = Qk - F(Qk) / F'(Qk) mod(x ^ pw)
    Qk = F;
  }
  Qk.resize(k);
  return Qk;
}
```

berlekamp-massey.hpp

866c28, 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;
}
```

botsan-mori.hpp

74e03a, 29 lines

```
// c - coefficients c[1], .., c[k] but 0-index
// a - initial values a[0], a[1], ..., a[k-1]
int botsanMori(VI c, VI a, LL n) {
  int k = SZ(c);
  assert(SZ(a) == k);
  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(k);
  while (n) {
    VI qMinus = q;
    for (int i = 1; i <= k; i += 2)
      qMinus[i] = sub(0, qMinus[i]);

    VI newP = mult(p, qMinus);
    VI newQ = mult(q, qMinus);
    FOR(i, 0, k)
      p[i] = newP[2 * i + (n & 1)];

    FOR(i, 0, k + 1)
      q[i] = newQ[2 * i];

    n >>= 1;
  }
  return mult(p[0], binPow(q[0], mod - 2));
}
```

7.1 Walsh-Hadamard

conv-xor.hpp

6a8bdc, 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] = u + v;
        a[j + (1 << i)] = 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-and.hpp

1614e9, 21 lines

```
void convAnd(VI& a, int k, bool inverse)
{
  FOR(i, 0, k)
    FOR(j, 0, 1 << k)
      if((j & (1 << i)) == 0)
      {
        if(inverse)
          a[j] -= a[j + (1 << i)];
        else
          a[j] += a[j + (1 << i)];
      }
}
VI multAnd(VI a, VI b, int k)
{
  convAnd(a, k, 0);
  convAnd(b, k, 0);
  FOR(i, 0, 1 << k)
    a[i] = mult(a[i], b[i]);
  convAnd(a, k, 1);
  return a;
}
```

conv-or.hpp

501340, 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)
          a[j + (1 << i)] -= a[j];
        else
          a[j + (1 << i)] += a[j];
      }
}
VI multOr(VI a, VI b, int k)
{
  convOr(a, k, 0);
  convOr(b, k, 0);
  FOR(i, 0, 1 << k)
    a[i] = mult(a[i], b[i]);
  convOr(a, k, 1);
  return a;
}
```

Binpow optimization

You can binpow in xor(and, or)-mult in $O(nlogn)$ by doing convolution in the start and in the end of binpow.

7.2 FFT with D and C

To calculate $g_{i+j} = \prod f_i * g_j$.
Use Divide and Conquer: when solve on $[l, r)$ for $l \leq i + j < r$ do $g_{i+j} += \prod_{l \leq j < r} f_i * g_j$.
Solve on $[l, m)$ after that update values in $[m, r)$ with values of g from $[l, m)$ and then solve on $[m, r)$.

Various (8)

mobius.hpp

fba6c5, 19 lines

```
void mobius()
{
  fill(pr, pr + N, 1);
  fill(mu, mu + N, 1);
  pr[1] = false;
  FOR (i, 2, N)
  {
    if (!pr[i])
      continue;
    mu[i] = mod - 1;
    for (int j = 2 * i; j < N; j += i)
    {
      pr[j] = false;
      if (j % (i * i) == 0)
        mu[j] = 0;
      mu[j] = mult(mu[j], mod - 1);
    }
  }
}
```

triangles.hpp

Description: finds all triangles in a graph. Should take vector of edges and **EMPTY** graph g. cnt++ respond to triangle v, u, w.
Time: $\mathcal{O}(m * sqrt(m))$

a22d8b, 30 lines

```
int triangles(int n, int m)
{
  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;
    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;
}
```

ternary.hpp

3db54c, 29 lines

```
const db phi = (3. - sqrt(5.0)) / 2.;
db get(db L, db R)
{
  db M1, M2, v1, v2;
  M1 = L + (R - L) * phi;
  M2 = R - (R - L) * phi;
  v1 = f(M1);
  v2 = f(M2);
  FOR (i, 0, 74)
  {
      if (v1 > v2) // for minimum
    {
      L = M1;
      M1 = M2;
      v1 = v2;
      M2 = R - (R - L) * phi;
      v2 = f(M2);
    }
    else
    {
      R = M2;
      M2 = M1;
      v2 = v1;
      M1 = L + (R - L) * phi;
      v1 = f(M1);
    }
  }
  return L; // or f(L);
}
```

8.1 Lader nim

Players have stone piles of size a0, a1, ..., an. In one move player can take 0 ¡ x ¡= ai stones from i-th pile and move them to (i-1)-th pile.

In this game you can forget about even piles. Take stones from odd is equal to remove in NIM and from even equal to add in NIM. Adding in NIM useless.

8.2 NP complete

Number of solutions to 2-SAT.

Formulas (9)

9.1 Modular formulas

9.1.1 $a^b\%m$

If $b >= phi(m)$ then b can be changed to $b\%phi(m) + phi(m)$.

9.1.2 Generators

Generator exist for $n = 1, 2, 4, p^k, 2p^k$ for odd primes p and positive integer k.

g is generator for modulo n if any comprime with n can be represented as $g^i, 0 \leq i < phi(n)$.

To find generator:

- find $phi(n)$ and $p_1, ..., p_m$ — prime factors of $phi(n)$
- g is generator only if $g^{\frac{phi(n)}{p_j}} \neq 1$ for each j
- check $g = 2, 3, 4, ..., p - 1$

9.1.3 Wilson

p is prime if and only if $(p-1)! = -1(modp)$.

9.1.4 Quadratic residue

q is called quadratic residue modulo n if there exist integer x that $x^2 = q(modn)$.

If n is odd prime than $\frac{p+1}{2}$ residues (including 0).

Legendre symbol (equal 0 if a divisible by p, 1 if residue and -1 if not):

$$(\frac{a}{p}) = a^{\frac{p-1}{2}}(modp)$$

Jacobi symbol (for composite n):

$$(\frac{a}{n}) = \prod (\frac{a}{p_i})^{\alpha_i}$$

9.2 Number Theory

9.2.1 Mobius

$$for n \geq 1 g(n) = \sum_{d|n} f(d)$$

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

$$M(n) = \sum_{k=1}^{n} \mu(k), \sum_{n=1} x M([x/n]) = 1$$

9.2.2 Catalan

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

$$C_n = \frac{1}{n+1} C_{2n}^{n}$$

$$C_n = C_{2n}^{n} - C_{2n} n - 1$$

9.2.3 Binomials

$$\sum_{k=0}^{n} C_n^k = 2^n$$

$$\sum_{m=0}^{n} C_m^k = C_{n+1}^{k+1}$$

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

$$\sum_{k=0}^{n} (C_n^k)^2 = C_{2n}^{n}$$

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

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

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

9.2.4 Fibonacci

$F_1 = F_2 = 1,\ F_n = F_{n-1} + F_{n-2}$
$F_{n+1} F_{n-1} - F_n^2 = (-1)^n,$
$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$
$gcd(F_m, F_n) = F_{gcd(n,m)}.$

$$F_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$$

9.2.5 Stirling

$S(n,k)$ — number of ways to divide n element into k non-empty groups.
$S(n,n) = 1,\ n \geq 0$
$S(n,0) = 0,\ n > 0$
$S(n,k) = S(n-1,k-1) + S(n-1,k) * k.$

$B_n = \sum S(n,k)$ from $n = 0$:

1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, 138295854,...

9.2.6 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 $G \cdot x$:

$$G \cdot x = \{g \cdot x \,|\, g \in G\}.$$

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, $X^g = \{x \in X \,|\, g \cdot x = x\}$. Burnside's lemma asserts the following formula for the number of orbits, denoted $|X/G|$:

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

9.3 Math Analysis

9.3.1 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \quad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x = 1 + \tan^2 x \quad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \quad \int x \sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x) \quad \int x e^{ax} dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

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

9.3.2 Series

$$f(x) = \sum_{n=0}^{\inf} \frac{f^{(n)}(a) * (x-a)^n}{n!}$$

$$e^x = \sum \frac{x^n}{n!}$$

$$\ln(1-x) = -\sum \frac{x^n}{n}, (-1 \leq x < 1)$$

9.3.3 Simpson

$$\int_a^b f(x) dx = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))$$

9.4 Numeric Analysis

9.4.1 Interpolations

Given unique pairs $(x_i, F(x_i))$, $0 \le i < n$
where $F(x)$ is polynomial which you must find.

Lagrange

$$l_i(x) = \prod_{j \ne i} \frac{x - x_j}{x_i - x_j}$$

$$F(x) = \sum F(x_i) * l_i(x)$$

If $x_i = i$, to calculate $F(X)$:
$P_i = \prod_{j<i}(X - x_j)$ and $S_i = \prod_{j>i}(X - x_j)$

$$l_i = \frac{P_i * S_i}{i! * (n-1-i)! * (-1)^{n-1-i}}$$

Newton

$$F_i(x) = F_{i-1}(x) + [y_0, \ldots, y_i](x - x_0)\ldots(x - x_i)$$

$$[y_i, y_i] = y_i, [y_l, y_r] = \frac{[y_{l+1}, y_r] - [y_l, y_{r-1}]}{x_r - x_l}$$

9.4.2 Runge-Kutta 4th Order Method for Ordinary Differential Equations

$$\frac{dy}{dx} = f(x, y), y(0) = y_0$$

$$x_{i+1} - x_i = h$$

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

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1h)$$

$$k_3 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2h)$$

$$k_4 = f(x_i + h, y_i + k_3h)$$

9.5 Geometry

9.5.1 Trigonometry

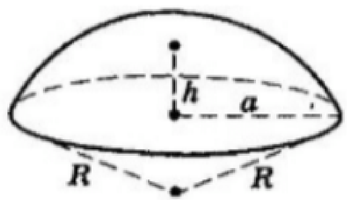
$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$\sin(v - w) = \sin v \cos w - \cos v \sin w$$
$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$

9.5.2 Triangles

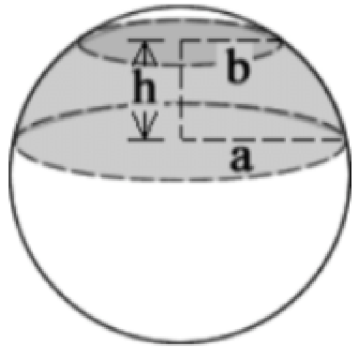
$S = \sqrt{p(p-a)(p-b)(p-c)}$, $S = \dfrac{abc}{4R} = pr$
Length of median: $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

9.5.3 Ball

$a = \sqrt{h * (2R - h)}$, $V = \pi * h^2(R - \frac{h}{3})$.



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



9.5.4 Ptolemey

$|AB| * |CD| + |BC| * |DA| \ge |AC| * |BD|$
Equality when ABCD on a circle.