Contents

1 Basic

1.1 Default code [f616de]

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
#define X first
#define Y second
#define SZ(a) ((int)a.size())
#define ALL(v) v.begin(), v.end()
#define pb push_back
#define eb emplace_back
```

1.2 Pragma [5feeb8]

```cpp
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,arch=skylake")
__builtin_ia32_ldmxcsr(
  __builtin_ia32_stmxcsr() | 0x8040)
```

1.3 readchar [dacef1]

```cpp
inline char readchar() {
  static const size_t bufsize = 65536;
  static char buf[bufsize];
  static char *p = buf, *end = buf;
  if (p == end)
    end = buf + fread_unlocked(buf, 1, bufsize, stdin),
    p = buf;
  return *p++;
}
```

1.4 debug [8f6825]

```cpp
void abc() { cerr << endl; }
template <typename T, typename... U>
void abc(T a, U... b) {
  cerr << a << ' ', abc(b...);
}
#ifdef debug
#define
     test(args...) abc("[" + string(#args) + "]", args)
#else
#define test(args...) void(0)
#endif
```

1.5 vimrc [dfda48]

```
se nu ai hls et ru ic is sc cul
se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
syntax on
hi cursorline cterm=none
"Select region and type :Hash to hash your selection."
ca Hash w !cpp -dD -P -fpreprocessed
     \| tr -d '[:space:]' \| md5sum \| cut -c-6
map <F9> :w !clear && g++ -
    std=c++17 -Ddebug -O2 -Wall -lm -g % && ./a.out<CR>
```

1.6 black magic [107dde]

```cpp
#include <ext/pb_ds/assoc_container.hpp> // rb_tree
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/rope> // rope
using namespace __gnu_pbds;
using namespace __gnu_cxx; // rope
typedef __gnu_pbds::priority_queue<int> heap;
int main() {
  heap h1, h2; // max heap
  h1.push(1), h1.push(3), h2.push(2), h2.push(4);
  h1.join(h2); // h1 = {1, 2, 3, 4}, h2 = {};
  tree<ll, null_type, less<ll>, rb_tree_tag,
    tree_order_statistics_node_update>
    st;
  tree<ll, ll, less<ll>, rb_tree_tag,
    tree_order_statistics_node_update>
    mp;
  for (int x : {0, 3, 20, 50}) st.insert(x);
  assert(st.order_of_key(3) == 1 &&
    st.order_of_key(4) == 2);
  assert(*st.find_by_order(2) == 20 &&
    *st.lower_bound(4) == 20);
  rope<char> *root[10]; // nsqrt(n)
  root[0] = new rope<char>();
  root[1] = new rope<char>(*root[0]);
  // root[1]->insert(pos, 'a');
  // root[1]->at(pos); 0-base
  // root[1]->erase(pos, size);
}
// __int128_t,__float128_t
// for (int i = bs._Find_first(); i < bs.size(); i =
// bs._Find_next(i));
```

2 Graph

2.1 SCC [517e91]

```cpp
struct SCC { // 0-base
  int n, dft, nscc;
  vector<int> low, dfn, bln, instack, stk;
  vector<vector<int>> G;
  void dfs(int u) {
    low[u] = dfn[u] = ++dft;
    instack[u] = 1, stk.pb(u);
    for (int v : G[u])
      if (!dfn[v])
        dfs(v), low[u] = min(low[u], low[v]);
      else if (instack[v] && dfn[v] < dfn[u])
        low[u] = min(low[u], dfn[v]);
    if (low[u] == dfn[u]) {
      for (; stk.back() != u; stk.pop_back())
        bln[stk.back()] = nscc,
        instack[stk.back()] = 0;
      instack[u] = 0, bln[u] = nscc++, stk.pop_back();
    }
  }
  SCC(int _n)
    : n(_n), dft(), nscc(), low(n), dfn(n), bln(n),
      instack(n), G(n) {}
  void add_edge(int u, int v) { G[u].pb(v); }
  void solve() {
    for (int i = 0; i < n; ++i)
      if (!dfn[i]) dfs(i);
  }
}; // scc_id(i): bln[i]
```

2.2 Minimum Arborescence [4c8d8d]

```cpp
struct zhu_liu { // O(VE)
  struct edge {
    int u, v;
    ll w;
  };
  vector<edge> E; // 0-base
  int pe[N], id[N], vis[N];
  ll in[N];
  void init() { E.clear(); }
  void add_edge(int u, int v, ll w) {
    if (u != v) E.pb(edge{u, v, w});
  }
  ll build(int root, int n) {
    ll ans = 0;
    for (;;) {
      fill_n(in, n, INF);
      for (int i = 0; i < SZ(E); ++i)
        if (E[i].u != E[i].v && E[i].w < in[E[i].v])
          pe[E[i].v] = i, in[E[i].v] = E[i].w;
      for (int u = 0; u < n; ++u) // no solution
        if (u != root && in[u] == INF) return -INF;
      int cntnode = 0;
      fill_n(id, n, -1), fill_n(vis, n, -1);
      for (int u = 0; u < n; ++u) {
        if (u != root) ans += in[u];
        int v = u;
        while (vis[v] != u && !~id[v] && v != root)
          vis[v] = u, v = E[pe[v]].u;
        if (v != root && !~id[v]) {
          for (int x = E[pe[v]].u; x != v;
               x = E[pe[x]].u)
            id[x] = cntnode;
          id[v] = cntnode++;
        }
      }
      if (!cntnode) break; // no cycle
      for (int u = 0; u < n; ++u)
        if (!~id[u]) id[u] = cntnode++;
```

```
      for (int i = 0; i < SZ(E); ++i) {
        int v = E[i].v;
        E[i].u = id[E[i].u], E[i].v = id[E[i].v];
        if (E[i].u != E[i].v) E[i].w -= in[v];
      }
      n = cntnode, root = id[root];
    }
    return ans;
  }
};
```

2.3 Dominator Tree [915f9c]

```
struct dominator_tree { // 1-base
  vector<int> G[N], rG[N];
  int n, pa[N], dfn[N], id[N], Time;
  int semi[N], idom[N], best[N];
  vector<int> tree[N]; // dominator_tree
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; ++i)
      G[i].clear(), rG[i].clear();
  }
  void add_edge(int u, int v) {
    G[u].pb(v), rG[v].pb(u);
  }
  void dfs(int u) {
    id[dfn[u] = ++Time] = u;
    for (auto v : G[u])
      if (!dfn[v]) dfs(v), pa[dfn[v]] = dfn[u];
  }
  int find(int y, int x) {
    if (y <= x) return y;
    int tmp = find(pa[y], x);
    if (semi[best[y]] > semi[best[pa[y]]])
      best[y] = best[pa[y]];
    return pa[y] = tmp;
  }
  void tarjan(int root) {
    Time = 0;
    for (int i = 1; i <= n; ++i) {
      dfn[i] = idom[i] = 0;
      tree[i].clear();
      best[i] = semi[i] = i;
    }
    dfs(root);
    for (int i = Time; i > 1; --i) {
      int u = id[i];
      for (auto v : rG[u])
        if (v = dfn[v]) {
          find(v, i);
          semi[i] = min(semi[i], semi[best[v]]);
        }
      tree[semi[i]].pb(i);
      for (auto v : tree[pa[i]]) {
        find(v, pa[i]);
        idom[v] =
          semi[best[v]] == pa[i] ? pa[i] : best[v];
      }
      tree[pa[i]].clear();
    }
    for (int i = 2; i <= Time; ++i) {
      if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
      tree[id[idom[i]]].pb(id[i]);
    }
  }
};
```

2.4 MinimumMeanCycle [e8ed41]

```
ll road[N][N]; // input here
struct MinimumMeanCycle {
  ll dp[N + 5][N], n;
  pll solve() {
    ll a = -1, b = -1, L = n + 1;
    for (int i = 2; i <= L; ++i)
      for (int k = 0; k < n; ++k)
        for (int j = 0; j < n; ++j)
          dp[i][j] =
            min(dp[i - 1][k] + road[k][j], dp[i][j]);
    for (int i = 0; i < n; ++i) {
      if (dp[L][i] >= INF) continue;
      ll ta = 0, tb = 1;
      for (int j = 1; j < n; ++j)
        if (dp[j][i] < INF &&
          ta * (L - j) < (dp[L][i] - dp[j][i]) * tb)
          ta = dp[L][i] - dp[j][i], tb = L - j;
      if (ta == 0) continue;
      if (a == -1 || a * tb > ta * b) a = ta, b = tb;
    }
    if (a != -1) {
      ll g = __gcd(a, b);
      return pll(a / g, b / g);
    }
    return pll(-1LL, -1LL);
  }
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) dp[i + 2][j] = INF;
  }
};
```

2.5 Minimum Clique Cover [745700]

```
struct Clique_Cover { // 0-base, O(n2^n)
  int co[1 << N], n, E[N];
  int dp[1 << N];
  void init(int _n) {
    n = _n, fill_n(dp, 1 << n, 0);
    fill_n(E, n, 0), fill_n(co, 1 << n, 0);
  }
  void add_edge(int u, int v) {
    E[u] |= 1 << v, E[v] |= 1 << u;
  }
  int solve() {
    for (int i = 0; i < n; ++i)
      co[1 << i] = E[i] | (1 << i);
    co[0] = (1 << n) - 1;
    dp[0] = (n & 1) * 2 - 1;
    for (int i = 1; i < (1 << n); ++i) {
      int t = i & -i;
      dp[i] = -dp[i ^ t];
      co[i] = co[i ^ t] & co[t];
    }
    for (int i = 0; i < (1 << n); ++i)
      co[i] = (co[i] & i) == i;
    fwt(co, 1 << n, 1);
    for (int ans = 1; ans < n; ++ans) {
      int sum = 0; // probabilistic
      for (int i = 0; i < (1 << n); ++i)
        sum += (dp[i] *= co[i]);
      if (sum) return ans;
    }
    return n;
  }
};
```

2.6 Maximum Clique Dyn [09472e]

```
struct MaxClique { // fast when N <= 100
  bitset<N> G[N], cs[N];
  int ans, sol[N], q, cur[N], d[N], n;
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].reset();
  }
  void add_edge(int u, int v) {
    G[u][v] = G[v][u] = 1;
  }
  void pre_dfs(vector<int> &r, int l, bitset<N> mask) {
    if (l < 4) {
      for (int i : r) d[i] = (G[i] & mask).count();
      sort(ALL(r),
        [&](int x, int y) { return d[x] > d[y]; });
    }
    vector<int> c(SZ(r));
    int lft = max(ans - q + 1, 1), rgt = 1, tp = 0;
    cs[1].reset(), cs[2].reset();
    for (int p : r) {
      int k = 1;
      while ((cs[k] & G[p]).any()) ++k;
      if (k > rgt) cs[++rgt + 1].reset();
      cs[k][p] = 1;
      if (k < lft) r[tp++] = p;
    }
    for (int k = lft; k <= rgt; ++k)
      for (int p = cs[k]._Find_first(); p < N;
           p = cs[k]._Find_next(p))
        r[tp] = p, c[tp] = k, ++tp;
    dfs(r, c, l + 1, mask);
  }
  void dfs(vector<int> &r, vector<int> &c, int l,
    bitset<N> mask) {
```

```
    while (!r.empty()) {
      int p = r.back();
      r.pop_back(), mask[p] = 0;
      if (q + c.back() <= ans) return;
      cur[q++] = p;
      vector<int> nr;
      for (int i : r)
        if (G[p][i]) nr.pb(i);
      if (!nr.empty()) pre_dfs(nr, l, mask & G[p]);
      else if (q > ans) ans = q, copy_n(cur, q, sol);
      c.pop_back(), --q;
    }
  }
  int solve() {
    vector<int> r(n);
    ans = q = 0, iota(ALL(r), 0);
    pre_dfs(r, 0, bitset<N>(string(n, '1')));
    return ans;
  }
};
```

2.7 Minimum Arborescence fast [48ee1b]

```
/* TODO
DSU: disjoint set
- DSU(n), .boss(x), .Union(x, y)
min_heap<T, Info>: min heap for type {T, Info} with
lazy tag
- .push({w, i}), .top(), .join(heap), .pop(), .empty(),
.add_lazy(v)
*/
struct E {
  int s, t;
  ll w;
}; // 0-base
vector<int> dmst(const vector<E> &e, int n, int root) {
  vector<min_heap<ll, int>> h(n * 2);
  for (int i = 0; i < SZ(e); ++i)
    h[e[i].t].push({e[i].w, i});
  DSU dsu(n * 2);
  vector<int> v(n * 2, -1), pa(n * 2, -1), r(n * 2);
  v[root] = n + 1;
  int pc = n;
  for (int i = 0; i < n; ++i)
    if (v[i] == -1) {
      for (int p = i; v[p] == -1 || v[p] == i;
           p = dsu.boss(e[r[p]].s)) {
        if (v[p] == i) {
          int q = p;
          p = pc++;
          do {
            h[q].add_lazy(-h[q].top().X);
            pa[q] = p, dsu.Union(p, q),
            h[p].join(h[q]);
          } while ((q = dsu.boss(e[r[q]].s)) != p);
        }
        v[p] = i;
        while (!h[p].empty() &&
          dsu.boss(e[h[p].top().Y].s) == p)
          h[p].pop();
        if (h[p].empty()) return {}; // no solution
        r[p] = h[p].top().Y;
      }
    }
  vector<int> ans;
  for (int i = pc - 1; i >= 0; i--)
    if (i != root && v[i] != n) {
      for (int f = e[r[i]].t; ~f && v[f] != n;
           f = pa[f])
        v[f] = n;
      ans.pb(r[i]);
    }
  return ans; // default minimize, returns edgeid array
} // O(Ef(E)), f(E) from min_heap
```

2.8 BCC Vertex [f56bab]

```
struct BCC { // 0-base
  int n, dft, nbcc;
  vector<int> low, dfn, bln, stk, is_ap, cir;
  vector<vector<int>> G, bcc, nG;
  void make_bcc(int u) {
    bcc.emplace_back(1, u);
    for (; stk.back() != u; stk.pop_back())
      bln[stk.back()] = nbcc, bcc[nbcc].pb(stk.back());
    stk.pop_back(), bln[u] = nbcc++;
  }
  void dfs(int u, int f) {
    int child = 0;
    low[u] = dfn[u] = ++dft, stk.pb(u);
    for (int v : G[u])
      if (!dfn[v]) {
        dfs(v, u), ++child;
        low[u] = min(low[u], low[v]);
        if (dfn[u] <= low[v]) {
          is_ap[u] = 1, bln[u] = nbcc;
          make_bcc(v), bcc.back().pb(u);
        }
      } else if (dfn[v] < dfn[u] && v != f)
        low[u] = min(low[u], dfn[v]);
    if (f == -1 && child < 2) is_ap[u] = 0;
    if (f == -1 && child == 0) make_bcc(u);
  }
  BCC(int _n)
    : n(_n), dft(), nbcc(), low(n), dfn(n), bln(n),
      is_ap(n), G(n) {}
  void add_edge(int u, int v) {
    G[u].pb(v), G[v].pb(u);
  }
  void solve() {
    for (int i = 0; i < n; ++i)
      if (!dfn[i]) dfs(i, -1);
  }
  void block_cut_tree() {
    cir.resize(nbcc);
    for (int i = 0; i < n; ++i)
      if (is_ap[i]) bln[i] = nbcc++;
    cir.resize(nbcc, 1), nG.resize(nbcc);
    for (int i = 0; i < nbcc && !cir[i]; ++i)
      for (int j : bcc[i])
        if (is_ap[j])
          nG[i].pb(bln[j]), nG[bln[j]].pb(i);
  } // up to 2 * n - 2 nodes!! bln[i] for id
};
```

2.9 NumberofMaximalClique [66fef5]

```
struct BronKerbosch { // 1-base
  int n, a[N], g[N][N];
  int S, all[N][N], some[N][N], none[N][N];
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j) g[i][j] = 0;
  }
  void add_edge(int u, int v) {
    g[u][v] = g[v][u] = 1;
  }
  void dfs(int d, int an, int sn, int nn) {
    if (S > 1000) return; // pruning
    if (sn == 0 && nn == 0) ++S;
    int u = some[d][0];
    for (int i = 0; i < sn; ++i) {
      int v = some[d][i];
      if (g[u][v]) continue;
      int tsn = 0, tnn = 0;
      copy_n(all[d], an, all[d + 1]);
      all[d + 1][an] = v;
      for (int j = 0; j < sn; ++j)
        if (g[v][some[d][j]])
          some[d + 1][tsn++] = some[d][j];
      for (int j = 0; j < nn; ++j)
        if (g[v][none[d][j]])
          none[d + 1][tnn++] = none[d][j];
      dfs(d + 1, an + 1, tsn, tnn);
      some[d][i] = 0, none[d][nn++] = v;
    }
  }
  int solve() {
    iota(some[0], some[0] + n, 1);
    S = 0, dfs(0, 0, n, 0);
    return S;
  }
};
```

2.10 2SAT [d0abc7]

```
struct SAT { // 0-base
  int n;
  vector<bool> istrue;
  SCC scc;
  SAT(int _n) : n(_n), istrue(n + n), scc(n + n) {}
  int rv(int a) { return a >= n ? a - n : a + n; }
  void add_clause(int a, int b) {
```

```
    scc.add_edge(rv(a), b), scc.add_edge(rv(b), a);
  }
  bool solve() {
    scc.solve();
    for (int i = 0; i < n; ++i) {
      if (scc.bln[i] == scc.bln[i + n]) return false;
      istrue[i] = scc.bln[i] < scc.bln[i + n];
      istrue[i + n] = !istrue[i];
    }
    return true;
  }
};
```

2.11 Virtual Tree [551777]

```
vector<int> vG[N];
int top, st[N];

void insert(int u) {
  if (top == -1) return st[++top] = u, void();
  int p = LCA(st[top], u);
  if (p == st[top]) return st[++top] = u, void();
  while (top >= 1 && dep[st[top - 1]] >= dep[p])
    vG[st[top - 1]].pb(st[top]), --top;
  if (st[top] != p)
    vG[p].pb(st[top]), --top, st[++top] = p;
  st[++top] = u;
}

void reset(int u) {
  for (int i : vG[u]) reset(i);
  vG[u].clear();
}

void solve(vector<int> &v) {
  top = -1;
  sort(ALL(v),
    [&](int a, int b) { return dfn[a] < dfn[b]; });
  for (int i : v) insert(i);
  while (top > 0) vG[st[top - 1]].pb(st[top]), --top;
  // do something
  reset(v[0]);
}
```

2.12 Bridge [f72ae7]

```
struct ECC { // 0-base
  int n, dft, ecnt, necc;
  vector<int> low, dfn, bln, is_bridge, stk;
  vector<vector<pii>> G;
  void dfs(int u, int f) {
    dfn[u] = low[u] = ++dft, stk.pb(u);
    for (auto [v, e] : G[u])
      if (!dfn[v])
        dfs(v, e), low[u] = min(low[u], low[v]);
      else if (e != f) low[u] = min(low[u], dfn[v]);
    if (low[u] == dfn[u]) {
      if (f != -1) is_bridge[f] = 1;
      for (; stk.back() != u; stk.pop_back())
        bln[stk.back()] = necc;
      bln[u] = necc++, stk.pop_back();
    }
  }
  ECC(int _n)
    : n(_n), dft(), ecnt(), necc(), low(n), dfn(n),
      bln(n), G(n) {}
  void add_edge(int u, int v) {
    G[u].pb(pii(v, ecnt)), G[v].pb(pii(u, ecnt++));
  }
  void solve() {
    is_bridge.resize(ecnt);
    for (int i = 0; i < n; ++i)
      if (!dfn[i]) dfs(i, -1);
  }
}; // ecc_id(i): bln[i]
```

2.13 MinimumSteinerTree [e6662f]

```
struct SteinerTree { // 0-base
  int n, dst[N][N], dp[1 << T][N], tdst[N];
  int vcst[N]; // the cost of vertexs
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) {
      fill_n(dst[i], n, INF);
      dst[i][i] = vcst[i] = 0;
    }
  }
  void chmin(int &x, int val) { x = min(x, val); }
  void add_edge(int ui, int vi, int wi) {
    chmin(dst[ui][vi], wi);
  }
  void shortest_path() {
    for (int k = 0; k < n; ++k)
      for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
          chmin(dst[i][j], dst[i][k] + dst[k][j]);
  }
  int solve(const vector<int> &ter) {
    shortest_path();
    int t = SZ(ter), full = (1 << t) - 1;
    for (int i = 0; i <= full; ++i)
      fill_n(dp[i], n, INF);
    copy_n(vcst, n, dp[0]);
    for (int msk = 1; msk <= full; ++msk) {
      if (!(msk & (msk - 1))) {
        int who = __lg(msk);
        for (int i = 0; i < n; ++i)
          dp[msk][i] =
            vcst[ter[who]] + dst[ter[who]][i];
      }
      for (int i = 0; i < n; ++i)
        for (int sub = (msk - 1) & msk; sub;
             sub = (sub - 1) & msk)
          chmin(dp[msk][i],
            dp[sub][i] + dp[msk ^ sub][i] - vcst[i]);
      for (int i = 0; i < n; ++i) {
        tdst[i] = INF;
        for (int j = 0; j < n; ++j)
          chmin(tdst[i], dp[msk][j] + dst[j][i]);
      }
      copy_n(tdst, n, dp[msk]);
    }
    return *min_element(dp[full], dp[full] + n);
  }
}; // O(V 3^T + V^2 2^T)
```

2.14 Vizing [2e3b51]

```
namespace vizing { // returns edge coloring in adjacent
                   // matrix G. 1 - based
const int N = 105;
int C[N][N], G[N][N], X[N], vst[N], n;
void init(int _n) {
  n = _n;
  for (int i = 0; i <= n; ++i)
    for (int j = 0; j <= n; ++j) C[i][j] = G[i][j] = 0;
}
void solve(vector<pii> &E) {
  auto update = [&](int u) {
    for (X[u] = 1; C[u][X[u]]; ++X[u]);
  };
  auto color = [&](int u, int v, int c) {
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v, C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if (p) X[u] = X[v] = p;
    else update(u), update(v);
    return p;
  };
  auto flip = [&](int u, int c1, int c2) {
    int p = C[u][c1];
    swap(C[u][c1], C[u][c2]);
    if (p) G[u][p] = G[p][u] = c2;
    if (!C[u][c1]) X[u] = c1;
    if (!C[u][c2]) X[u] = c2;
    return p;
  };
  fill_n(X + 1, n, 1);
  for (int t = 0; t < SZ(E); ++t) {
    int u = E[t].X, v0 = E[t].Y, v = v0, c0 = X[u],
        c = c0, d;
    vector<pii> L;
    fill_n(vst + 1, n, 0);
    while (!G[u][v0]) {
      L.emplace_back(v, d = X[v]);
      if (!C[v][c])
        for (int a = SZ(L) - 1; a >= 0; --a)
          c = color(u, L[a].X, c);
      else if (!C[u][d])
        for (int a = SZ(L) - 1; a >= 0; --a)
          color(u, L[a].X, L[a].Y);
      else if (vst[d]) break;
```

```
      else vst[d] = 1, v = C[u][d];
    }
    if (!G[u][v0]) {
      for (; v; v = flip(v, c, d), swap(c, d));
      if (int a; C[u][c0]) {
        for (a = SZ(L) - 2; a >= 0 && L[a].Y != c;
             --a);
        for (; a >= 0; --a) color(u, L[a].X, L[a].Y);
      } else --t;
    }
  }
}
} // namespace vizing
```

2.15 Maximum Clique [03ff71]

```
struct Maximum_Clique {
  typedef bitset<MAXN> bst;
  bst N[MAXN], empty;
  int p[MAXN], n, ans;
  void BronKerbosch2(bst R, bst P, bst X) {
    if (P == empty && X == empty)
      return ans = max(ans, (int)R.count()), void();
    bst tmp = P | X;
    int u;
    if ((R | P | X).count() <= ans) return;
    for (int uu = 0; uu < n; ++uu) {
      u = p[uu];
      if (tmp[u] == 1) break;
    }
    // if (double(clock())/CLOCKS_PER_SEC > .999)
    // return;
    bst now2 = P & ~N[u];
    for (int vv = 0; vv < n; ++vv) {
      int v = p[vv];
      if (now2[v] == 1) {
        R[v] = 1;
        BronKerbosch2(R, P & N[v], X & N[v]);
        R[v] = 0, P[v] = 0, X[v] = 1;
      }
    }
  }
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) N[i].reset();
  }
  void add_edge(int u, int v) {
    N[u][v] = N[v][u] = 1;
  }
  int solve() { // remember srand
    bst R, P, X;
    ans = 0, P.flip();
    for (int i = 0; i < n; ++i) p[i] = i;
    random_shuffle(p, p + n), BronKerbosch2(R, P, X);
    return ans;
  }
};
```

3 Data Structure

3.1 2D Segment Tree [6985fc]

```
int num[501][501], N, M; // input here
struct seg_2D {
  struct node {
    int data;
    node *lc, *rc;
  } *root;
  node *merge(node *a, node *b, int l, int r) {
    node *p = new node;
    p->data = max(a->data, b->data);
    if (l == r) return p;
    int m = l + r >> 1;
    p->lc = merge(a->lc, b->lc, l, m);
    p->rc = merge(a->rc, b->rc, m + 1, r);
    return p;
  }
  node *build(int l, int r, int x) {
    node *p = new node;
    if (l == r) return p->data = num[x][l], p;
    int m = l + r >> 1;
    p->lc = build(l, m, x), p->rc = build(m + 1, r, x);
    p->data = max(p->lc->data, p->rc->data);
    return p;
  }
  int query(int L, int R, int l, int r, node *p) {
    if (L <= l && R >= r) return p->data;
    int m = l + r >> 1, re = 0;
    if (L <= m) re = query(L, R, l, m, p->lc);
    if (R > m)
      re = max(re, query(L, R, m + 1, r, p->rc));
    return re;
  }
};
struct seg_1D {
  struct node {
    seg_2D data;
    node *lc, *rc;
  } *root;
  node *s_build(int l, int r) {
    node *p = new node;
    if (l == r)
      return p->data.root = p->data.build(1, M, l), p;
    int m = l + r >> 1;
    p->lc = s_build(l, m), p->rc = s_build(m + 1, r);
    p->data.root = p->data.merge(
      p->lc->data.root, p->rc->data.root, 1, M);
    return p;
  }
  int s_query(int L, int R, int l, int r, node *p,
    int yl, int yr) {
    if (L <= l && R >= r)
      return p->data.query(yl, yr, 1, M, p->data.root);
    int m = l + r >> 1, re = 0;
    if (L <= m)
      re = s_query(L, R, l, m, p->lc, yl, yr);
    if (R > m)
      re = max(
        re, s_query(L, R, m + 1, r, p->rc, yl, yr));
    return re;
  }
  void init() { root = s_build(1, N); }
  int query(int xl, int xr, int yl, int yr) {
    return s_query(xl, xr, 1, N, root, yl, yr);
  }
};
```

3.2 Sparse table [cef484]

```
struct Sparse_table {
  int st[__lg(MAXN) + 1][MAXN], n;
  void init(int _n, int *data) {
    n = _n;
    for (int i = 0; i < n; ++i) st[0][i] = data[i];
    for (int i = 1, t = 2; t < n; t <<= 1, i++)
      for (int j = 0; j + t <= n; j++)
        st[i][j] =
          max(st[i - 1][j], st[i - 1][j + t / 2]);
  }
  int query(int a, int b) {
    int t = __lg(b - a + 1);
    return max(st[t][a], st[t][b - (1 << t) + 1]);
  }
};
```

3.3 Binary Index Tree [18be78]

```
struct Binary_Index_Tree {
  int bit[MAXN + 1], lazy[MAXN + 1], n;
  int lb(int x) { return x & -x; }
  void init(int _n, int *data) {
    n = _n;
    for (int i = 1, t; i <= n; ++i) {
      bit[i] = data[i], lazy[i] = 0, t = i - lb(i);
      for (int j = i - 1; j > t; j -= lb(j))
        bit[i] += bit[j];
    }
  }
  void suf_modify(int x, int v) {
    for (int t = x; t; t -= lb(t)) lazy[t] += v;
    for (int t = x + lb(x); t && t <= n; t += lb(t))
      bit[t] += v * (x - t + lb(t));
  }
  void modify(int x, int v) {
    for (; x; x -= lb(x)) bit[x] += v;
  }
  int query(int x) {
    int re = 0;
    for (int t = x; t; t -= lb(t))
      re += lazy[t] * lb(t) + bit[t];
    for (int t = x + lb(x); t && t <= n; t += lb(t))
      re += lazy[t] * (x - t + lb(t));
    return re;
  }
};
```

3.4 Segment Tree [0f243e]

```
struct Segment_Tree {
  struct node {
    int data, lazy;
    node *l, *r;
    node() : data(0), lazy(0), l(0), r(0) {}
    void up() {
      if (l) data = max(l->data, r->data);
    }
    void down() {
      if (l) {
        l->data += lazy, l->lazy += lazy;
        r->data += lazy, r->lazy += lazy;
      }
      lazy = 0;
    }
  } *root;
  int l, r;
  node *build(int l, int r, int *data) {
    node *p = new node();
    if (l == r) return p->data = data[l], p;
    int m = (l + r) / 2;
    p->l = build(l, m, data),
    p->r = build(m + 1, r, data);
    return p->up(), p;
  }
  void s_modify(
    int L, int R, int l, int r, node *p, int x) {
    if (r < L || l > R) return;
    p->down();
    if (L <= l && R >= r)
      return p->data += x, p->lazy += x, void();
    int m = (l + r) / 2;
    s_modify(L, R, l, m, p->l, x);
    s_modify(L, R, m + 1, r, p->r, x);
    p->up();
  }
  int s_query(int L, int R, int l, int r, node *p) {
    p->down();
    if (L <= l && R >= r) return p->data;
    int m = (l + r) / 2;
    if (R <= m) return s_query(L, R, l, m, p->l);
    if (L > m) return s_query(L, R, m + 1, r, p->r);
    return max(s_query(L, R, l, m, p->l),
      s_query(L, R, m + 1, r, p->r));
  }
  void init(int L, int R, int *data) {
    l = L, r = R;
    root = build(l, r, data);
  }
  void modify(int L, int R, int x) {
    s_modify(L, R, l, r, root, x);
  }
  int query(int L, int R) {
    return s_query(L, R, l, r, root);
  }
};
```

3.5 BIT kth [7de9a0]

```
int bit[N + 1]; // N = 2 ^ k
int query_kth(int k) {
  int res = 0;
  for (int i = N >> 1; i >= 1; i >>= 1)
    if (bit[res + i] < k) k -= bit[res += i];
  return res + 1;
}
```

3.6 Centroid Decomposition [6971c7]

```
struct Cent_Dec { // 1-base
  vector<pll> G[N];
  pll info[N]; // store info. of itself
  pll upinfo[N]; // store info. of climbing up
  int n, pa[N], layer[N], sz[N], done[N];
  ll dis[__lg(N) + 1][N];
  void init(int _n) {
    n = _n, layer[0] = -1;
    fill_n(pa + 1, n, 0), fill_n(done + 1, n, 0);
    for (int i = 1; i <= n; ++i) G[i].clear();
  }
  void add_edge(int a, int b, int w) {
    G[a].pb(pll(b, w)), G[b].pb(pll(a, w));
  }
  void get_cent(
    int u, int f, int &mx, int &c, int num) {
    int mxsz = 0;
    sz[u] = 1;
    for (pll e : G[u])
      if (!done[e.X] && e.X != f) {
        get_cent(e.X, u, mx, c, num);
        sz[u] += sz[e.X], mxsz = max(mxsz, sz[e.X]);
      }
    if (mx > max(mxsz, num - sz[u]))
      mx = max(mxsz, num - sz[u]), c = u;
  }
  void dfs(int u, int f, ll d, int org) {
    // if required, add self info or climbing info
    dis[layer[org]][u] = d;
    for (pll e : G[u])
      if (!done[e.X] && e.X != f)
        dfs(e.X, u, d + e.Y, org);
  }
  int cut(int u, int f, int num) {
    int mx = 1e9, c = 0, lc;
    get_cent(u, f, mx, c, num);
    done[c] = 1, pa[c] = f, layer[c] = layer[f] + 1;
    for (pll e : G[c])
      if (!done[e.X]) {
        if (sz[e.X] > sz[c])
          lc = cut(e.X, c, num - sz[c]);
        else lc = cut(e.X, c, sz[e.X]);
        upinfo[lc] = pll(), dfs(e.X, c, e.Y, c);
      }
    return done[c] = 0, c;
  }
  void build() { cut(1, 0, n); }
  void modify(int u) {
    for (int a = u, ly = layer[a]; a;
         a = pa[a], --ly) {
      info[a].X += dis[ly][u], ++info[a].Y;
      if (pa[a])
        upinfo[a].X += dis[ly - 1][u], ++upinfo[a].Y;
    }
  }
  ll query(int u) {
    ll rt = 0;
    for (int a = u, ly = layer[a]; a;
         a = pa[a], --ly) {
      rt += info[a].X + info[a].Y * dis[ly][u];
      if (pa[a])
        rt -=
          upinfo[a].X + upinfo[a].Y * dis[ly - 1][u];
    }
    return rt;
  }
};
```

3.7 DSU [e8502d]

```
struct DSU {
  vector<int> arr;
  DSU(int n = 0) : arr(n) { iota(ALL(arr), 0); }
  int boss(int x) {
    if (arr[x] == x) return x;
    return arr[x] = boss(arr[x]);
  }
  bool Union(int x, int y) {
    x = boss(x), y = boss(y);
    if (x == y) return 0;
    arr[y] = x;
    return 1;
  }
};
```

3.8 Smart Pointer [7f0fff]

```
#ifndef REFERENCE_POINTER
#define REFERENCE_POINTER
template <typename T> struct _RefCounter {
  T data;
  int ref;
  _RefCounter(const T &d = 0) : data(d), ref(0) {}
};
template <typename T> struct reference_pointer {
  _RefCounter<T> *p;
  T *operator->() { return &p->data; }
  T &operator*() { return p->data; }
  operator _RefCounter<T> *() { return p; }
  reference_pointer &operator=(
    const reference_pointer &t) {
    if (p && !--p->ref) delete p;
    p = t.p;
```

```
    p && ++p->ref;
    return *this;
  }
  reference_pointer(_RefCounter<T> *t = 0) : p(t) {
    p && ++p->ref;
  }
  reference_pointer(const reference_pointer &t)
    : p(t.p) {
    p && ++p->ref;
  }
  ~reference_pointer() {
    if (p && !--p->ref) delete p;
  }
};
template <typename T>
inline reference_pointer<T> new_reference(
  const T &nd) {
  return reference_pointer<T>(new _RefCounter<T>(nd));
}
#endif
// note:
reference_pointer<int> a;
a = new_reference(5);
a = new_reference<int>(5);
a = new_reference((int)5);
reference_pointer<int> b = a;

struct P {
  int a, b;
  P(int _a, int _b) : a(_a), b(_b) {}
} p(2, 3);
reference_pointer<P> a;
c = new_reference(P(1, 2));
c = new_reference<P>(P(1, 2));
c = new_reference(p);
```

3.9 IntervalContainer [dbcccd]

```
/* Add and remove intervals from a set of disjoint
 * intervals. Will merge the added interval with any
 * overlapping intervals in the set when adding.
 * Intervals are [inclusive, exclusive). */
set<pii>::iterator addInterval(
  set<pii> &is, int L, int R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->X <= R) {
    R = max(R, it->Y);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->Y >= L) {
    L = min(L, it->X);
    R = max(R, it->Y);
    is.erase(it);
  }
  return is.insert(before, pii(L, R));
}
void removeInterval(set<pii> &is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->Y;
  if (it->X == L) is.erase(it);
  else (int &)it->Y = L;
  if (R != r2) is.emplace(R, r2);
}
```

3.10 KDTree useful [22a1d3]

```
template <typename T, size_t kd> // kd???????
class kd_tree {
public:
  struct point {
    T d[kd];
    inline T dist(const point &x) const {
      T ret = 0;
      for (size_t i = 0; i < kd; ++i)
        ret += std::abs(d[i] - x.d[i]);
      return ret;
    }
    inline bool operator==(const point &p) {
      for (size_t i = 0; i < kd; ++i) {
        if (d[i] != p.d[i]) return 0;
      }
      return 1;
    }
    inline bool operator<(const point &b) const {
      return d[0] < b.d[0];
    }
  };

private:
  struct node {
    node *l, *r;
    point pid;
    int s;
    node(const point &p) : l(0), r(0), pid(p), s(1) {}
    inline void up() {
      s = (l ? l->s : 0) + 1 + (r ? r->s : 0);
    }
  } *root;
  const double alpha, loga;
  const T INF; //????INF,?????
  int maxn;
  struct __cmp {
    int sort_id;
    inline bool operator()(
      const node *x, const node *y) const {
      return operator()(x->pid, y->pid);
    }
    inline bool operator()(
      const point &x, const point &y) const {
      if (x.d[sort_id] != y.d[sort_id])
        return x.d[sort_id] < y.d[sort_id];
      for (size_t i = 0; i < kd; ++i) {
        if (x.d[i] != y.d[i]) return x.d[i] < y.d[i];
      }
      return 0;
    }
  } cmp;
  void clear(node *o) {
    if (!o) return;
    clear(o->l);
    clear(o->r);
    delete o;
  }
  inline int size(node *o) { return o ? o->s : 0; }
  std::vector<node *> A;
  node *build(int k, int l, int r) {
    if (l > r) return 0;
    if (k == kd) k = 0;
    int mid = (l + r) / 2;
    cmp.sort_id = k;
    std::nth_element(A.begin() + l, A.begin() + mid,
      A.begin() + r + 1, cmp);
    node *ret = A[mid];
    ret->l = build(k + 1, l, mid - 1);
    ret->r = build(k + 1, mid + 1, r);
    ret->up();
    return ret;
  }
  inline bool isbad(node *o) {
    return size(o->l) > alpha * o->s ||
      size(o->r) > alpha * o->s;
  }
  void flatten(node *u,
    typename std::vector<node *>::iterator &it) {
    if (!u) return;
    flatten(u->l, it);
    *it = u;
    flatten(u->r, ++it);
  }
  inline void rebuild(node *&u, int k) {
    if ((int)A.size() < u->s) A.resize(u->s);
    typename std::vector<node *>::iterator it =
      A.begin();
    flatten(u, it);
    u = build(k, 0, u->s - 1);
  }
  bool insert(
    node *&u, int k, const point &x, int dep) {
    if (!u) {
      u = new node(x);
      return dep <= 0;
    }
    ++u->s;
    cmp.sort_id = k;
    if (insert(cmp(x, u->pid) ? u->l : u->r,
          (k + 1) % kd, x, dep - 1)) {
      if (!isbad(u)) return 1;
      rebuild(u, k);
    }
    return 0;
  }
```

```
  node *findmin(node *o, int k) {
    if (!o) return 0;
    if (cmp.sort_id == k)
      return o->l ? findmin(o->l, (k + 1) % kd) : o;
    node *l = findmin(o->l, (k + 1) % kd);
    node *r = findmin(o->r, (k + 1) % kd);
    if (l && !r) return cmp(l, o) ? l : o;
    if (!l && r) return cmp(r, o) ? r : o;
    if (!l && !r) return o;
    if (cmp(l, r)) return cmp(l, o) ? l : o;
    return cmp(r, o) ? r : o;
  }
  bool erase(node *&u, int k, const point &x) {
    if (!u) return 0;
    if (u->pid == x) {
      if (u->r)
        ;
      else if (u->l) {
        u->r = u->l;
        u->l = 0;
      } else {
        delete u;
        u = 0;
        return 1;
      }
      --u->s;
      cmp.sort_id = k;
      u->pid = findmin(u->r, (k + 1) % kd)->pid;
      return erase(u->r, (k + 1) % kd, u->pid);
    }
    cmp.sort_id = k;
    if (erase(cmp(x, u->pid) ? u->l : u->r,
          (k + 1) % kd, x)) {
      --u->s;
      return 1;
    } else return 0;
  }
  inline T heuristic(const T h[]) const {
    T ret = 0;
    for (size_t i = 0; i < kd; ++i) ret += h[i];
    return ret;
  }
  int qM;
  std::priority_queue<std::pair<T, point>> pQ;
  void nearest(
    node *u, int k, const point &x, T *h, T &mndist) {
    if (u == 0 || heuristic(h) >= mndist) return;
    T dist = u->pid.dist(x), old = h[k];
    /*mndist=std::min(mndist,dist);*/
    if (dist < mndist) {
      pQ.push(std::make_pair(dist, u->pid));
      if ((int)pQ.size() == qM + 1) {
        mndist = pQ.top().first, pQ.pop();
      }
    }
    if (x.d[k] < u->pid.d[k]) {
      nearest(u->l, (k + 1) % kd, x, h, mndist);
      h[k] = std::abs(x.d[k] - u->pid.d[k]);
      nearest(u->r, (k + 1) % kd, x, h, mndist);
    } else {
      nearest(u->r, (k + 1) % kd, x, h, mndist);
      h[k] = std::abs(x.d[k] - u->pid.d[k]);
      nearest(u->l, (k + 1) % kd, x, h, mndist);
    }
    h[k] = old;
  }
  std::vector<point> in_range;
  void range(
    node *u, int k, const point &mi, const point &ma) {
    if (!u) return;
    bool is = 1;
    for (int i = 0; i < kd; ++i)
      if (u->pid.d[i] < mi.d[i] ||
        ma.d[i] < u->pid.d[i]) {
        is = 0;
        break;
      }
    if (is) in_range.push_back(u->pid);
    if (mi.d[k] <= u->pid.d[k])
      range(u->l, (k + 1) % kd, mi, ma);
    if (ma.d[k] >= u->pid.d[k])
      range(u->r, (k + 1) % kd, mi, ma);
  }

public:
  kd_tree(const T &INF, double a = 0.75)
    : root(0), alpha(a), loga(log2(1.0 / a)), INF(INF),
      maxn(1) {}
  inline void clear() {
    clear(root), root = 0, maxn = 1;
  }
  inline void build(int n, const point *p) {
    clear(root), A.resize(maxn = n);
    for (int i = 0; i < n; ++i) A[i] = new node(p[i]);
    root = build(0, 0, n - 1);
  }
  inline void insert(const point &x) {
    insert(root, 0, x, std::__lg(size(root)) / loga);
    if (root->s > maxn) maxn = root->s;
  }
  inline bool erase(const point &p) {
    bool d = erase(root, 0, p);
    if (root && root->s < alpha * maxn) rebuild();
    return d;
  }
  inline void rebuild() {
    if (root) rebuild(root, 0);
    maxn = root->s;
  }
  inline T nearest(const point &x, int k) {
    qM = k;
    T mndist = INF, h[kd] = {};
    nearest(root, 0, x, h, mndist);
    mndist = pQ.top().first;
    pQ = std::priority_queue<std::pair<T, point>>();
    return mndist; /*???x?k??????*/
  }
  inline const std::vector<point> &range(
    const point &mi, const point &ma) {
    in_range.clear();
    range(root, 0, mi, ma);
    return in_range; /*????mi?ma????vector*/
  }
  inline int size() { return root ? root->s : 0; }
};
```

3.11 min heap [b3de3d]

```
template <class T, class Info> struct min_heap {
  priority_queue<pair<T, Info>, vector<pair<T, Info>>,
    greater<pair<T, Info>>>
    pq;
  T lazy = 0;
  void push(pair<T, Info> v) {
    pq.emplace(v.X - lazy, v.Y);
  }
  pair<T, Info> top() {
    return make_pair(pq.top().X + lazy, pq.top().Y);
  }
  void join(min_heap &rgt) {
    if (SZ(pq) < SZ(rgt.pq)) {
      swap(pq, rgt.pq);
      swap(lazy, rgt.lazy);
    }
    while (!rgt.pq.empty()) {
      push(rgt.top());
      rgt.pop();
    }
  }
  void pop() { pq.pop(); }
  bool empty() { return pq.empty(); }
  void add_lazy(T v) { lazy += v; }
};
```

3.12 LiChaoST [2c55c3]

```
struct L {
  ll m, k, id;
  L() : id(-1) {}
  L(ll a, ll b, ll c) : m(a), k(b), id(c) {}
  ll at(ll x) { return m * x + k; }
};
class LiChao { // maintain max
private:
  int n;
  vector<L> nodes;
  void insert(int l, int r, int rt, L ln) {
    int m = (l + r) >> 1;
    if (nodes[rt].id == -1)
      return nodes[rt] = ln, void();
    bool atLeft = nodes[rt].at(l) < ln.at(l);
    if (nodes[rt].at(m) < ln.at(m))
      atLeft ^= 1, swap(nodes[rt], ln);
```

```
    if (r - l == 1) return;
    if (atLeft) insert(l, m, rt << 1, ln);
    else insert(m, r, rt << 1 | 1, ln);
  }
  ll query(int l, int r, int rt, ll x) {
    int m = (l + r) >> 1;
    ll ret = -INF;
    if (nodes[rt].id != -1) ret = nodes[rt].at(x);
    if (r - l == 1) return ret;
    if (x < m)
      return max(ret, query(l, m, rt << 1, x));
    return max(ret, query(m, r, rt << 1 | 1, x));
  }

public:
  LiChao(int n_) : n(n_), nodes(n * 4) {}
  void insert(L ln) { insert(0, n, 1, ln); }
  ll query(ll x) { return query(0, n, 1, x); }
};
```

3.13 Treap [4a5ee3]

```
struct node {
  int data, sz;
  node *l, *r;
  node(int k) : data(k), sz(1), l(0), r(0) {}
  void up() {
    sz = 1;
    if (l) sz += l->sz;
    if (r) sz += r->sz;
  }
  void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
  if (!a || !b) return a ? a : b;
  if (rand() % (sz(a) + sz(b)) < sz(a))
    return a->down(), a->r = merge(a->r, b), a->up(),
                      a;
  return b->down(), b->l = merge(a, b->l), b->up(), b;
}
void split(node *o, node *&a, node *&b, int k) {
  if (!o) return a = b = 0, void();
  o->down();
  if (o->data <= k)
    a = o, split(o->r, a->r, b, k), a->up();
  else b = o, split(o->l, a, b->l, k), b->up();
}
void split2(node *o, node *&a, node *&b, int k) {
  if (sz(o) <= k) return a = o, b = 0, void();
  o->down();
  if (sz(o->l) + 1 <= k)
    a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
  else b = o, split2(o->l, a, b->l, k);
  o->up();
}
node *kth(node *o, int k) {
  if (k <= sz(o->l)) return kth(o->l, k);
  if (k == sz(o->l) + 1) return o;
  return kth(o->r, k - sz(o->l) - 1);
}
int Rank(node *o, int key) {
  if (!o) return 0;
  if (o->data < key)
    return sz(o->l) + 1 + Rank(o->r, key);
  else return Rank(o->l, key);
}
bool erase(node *&o, int k) {
  if (!o) return 0;
  if (o->data == k) {
    node *t = o;
    o->down(), o = merge(o->l, o->r);
    delete t;
    return 1;
  }
  node *&t = k < o->data ? o->l : o->r;
  return erase(t, k) ? o->up(), 1 : 0;
}
void insert(node *&o, int k) {
  node *a, *b;
  split(o, a, b, k),
    o = merge(a, merge(new node(k), b));
}
void interval(node *&o, int l, int r) {
  node *a, *b, *c;
  split2(o, a, b, l - 1), split2(b, b, c, r);
  // operate
  o = merge(a, merge(b, c));
}
```

3.14 link cut tree [703f02]

```
struct Splay { // xor-sum
  static Splay nil;
  Splay *ch[2], *f;
  int val, sum, rev, size;
  Splay(int _val = 0)
    : val(_val), sum(_val), rev(0), size(1) {
    f = ch[0] = ch[1] = &nil;
  }
  bool isr() {
    return f->ch[0] != this && f->ch[1] != this;
  }
  int dir() { return f->ch[0] == this ? 0 : 1; }
  void setCh(Splay *c, int d) {
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void give_tag(int r) {
    if (r) swap(ch[0], ch[1]), rev ^= 1;
  }
  void push() {
    if (ch[0] != &nil) ch[0]->give_tag(rev);
    if (ch[1] != &nil) ch[1]->give_tag(rev);
    rev = 0;
  }
  void pull() {
    // take care of the nil!
    size = ch[0]->size + ch[1]->size + 1;
    sum = ch[0]->sum ^ ch[1]->sum ^ val;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil;
Splay *nil = &Splay::nil;
void rotate(Splay *x) {
  Splay *p = x->f;
  int d = x->dir();
  if (!p->isr()) p->f->setCh(x, p->dir());
  else x->f = p->f;
  p->setCh(x->ch[!d], d);
  x->setCh(p, !d);
  p->pull(), x->pull();
}
void splay(Splay *x) {
  vector<Splay *> splayVec;
  for (Splay *q = x;; q = q->f) {
    splayVec.pb(q);
    if (q->isr()) break;
  }
  reverse(ALL(splayVec));
  for (auto it : splayVec) it->push();
  while (!x->isr()) {
    if (x->f->isr()) rotate(x);
    else if (x->dir() == x->f->dir())
      rotate(x->f), rotate(x);
    else rotate(x), rotate(x);
  }
}
Splay *access(Splay *x) {
  Splay *q = nil;
  for (; x != nil; x = x->f)
    splay(x), x->setCh(q, 1), q = x;
  return q;
}
void root_path(Splay *x) { access(x), splay(x); }
void chroot(Splay *x) {
  root_path(x), x->give_tag(1);
  x->push(), x->pull();
}
void split(Splay *x, Splay *y) {
  chroot(x), root_path(y);
}
void link(Splay *x, Splay *y) {
  root_path(x), chroot(y);
  x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
  split(x, y);
  if (y->size != 5) return;
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}
```

```
Splay *get_root(Splay *x) {
  for (root_path(x); x->ch[0] != nil; x = x->ch[0])
    x->push();
  splay(x);
  return x;
}
bool conn(Splay *x, Splay *y) {
  return get_root(x) == get_root(y);
}
Splay *lca(Splay *x, Splay *y) {
  access(x), root_path(y);
  if (y->f == nil) return y;
  return y->f;
}
void change(Splay *x, int val) {
  splay(x), x->val = val, x->pull();
}
int query(Splay *x, Splay *y) {
  split(x, y);
  return y->sum;
}
```

3.15 Heavy light Decomposition [b91cf9]

```
struct Heavy_light_Decomposition { // 1-base
  int n, ulink[N], deep[N], mxson[N], w[N], pa[N];
  int t, pl[N], data[N], val[N]; // val: vertex data
  vector<int> G[N];
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; ++i)
      G[i].clear(), mxson[i] = 0;
  }
  void add_edge(int a, int b) {
    G[a].pb(b), G[b].pb(a);
  }
  void dfs(int u, int f, int d) {
    w[u] = 1, pa[u] = f, deep[u] = d++;
    for (int &i : G[u])
      if (i != f) {
        dfs(i, u, d), w[u] += w[i];
        if (w[mxson[u]] < w[i]) mxson[u] = i;
      }
  }
  void cut(int u, int link) {
    data[pl[u] = ++t] = val[u], ulink[u] = link;
    if (!mxson[u]) return;
    cut(mxson[u], link);
    for (int i : G[u])
      if (i != pa[u] && i != mxson[u]) cut(i, i);
  }
  void build() { dfs(1, 1, 1), cut(1, 1), /*build*/; }
  int query(int a, int b) {
    int ta = ulink[a], tb = ulink[b], res = 0;
    while (ta != tb) {
      if (deep[ta] > deep[tb])
        swap(ta, tb), swap(a, b);
      // query(pl[tb], pl[b])
      tb = ulink[b = pa[tb]];
    }
    if (pl[a] > pl[b]) swap(a, b);
    // query(pl[a], pl[b])
  }
};
```

3.16 Leftist Tree [2201dc]

```
struct node {
  ll v, data, sz, sum;
  node *l, *r;
  node(ll k)
    : v(0), data(k), sz(1), l(0), r(0), sum(k) {}
};
ll sz(node *p) { return p ? p->sz : 0; }
ll V(node *p) { return p ? p->v : -1; }
ll sum(node *p) { return p ? p->sum : 0; }
node *merge(node *a, node *b) {
  if (!a || !b) return a ? a : b;
  if (a->data < b->data) swap(a, b);
  a->r = merge(a->r, b);
  if (V(a->r) > V(a->l)) swap(a->r, a->l);
  a->v = V(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
  a->sum = sum(a->l) + sum(a->r) + a->data;
  return a;
}
void pop(node *&o) {
  node *tmp = o;
  o = merge(o->l, o->r);
  delete tmp;
}
```

3.17 KDTree [85f231]

```
namespace kdt {
int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn],
  yl[maxn], yr[maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
  if (l == r) return -1;
  function<bool(const point &, const point &)> f =
    [dep](const point &a, const point &b) {
      if (dep & 1) return a.x < b.x;
      else return a.y < b.y;
    };
  int m = (l + r) >> 1;
  nth_element(p + l, p + m, p + r, f);
  xl[m] = xr[m] = p[m].x;
  yl[m] = yr[m] = p[m].y;
  lc[m] = build(l, m, dep + 1);
  if (~lc[m]) {
    xl[m] = min(xl[m], xl[lc[m]]);
    xr[m] = max(xr[m], xr[lc[m]]);
    yl[m] = min(yl[m], yl[lc[m]]);
    yr[m] = max(yr[m], yr[lc[m]]);
  }
  rc[m] = build(m + 1, r, dep + 1);
  if (~rc[m]) {
    xl[m] = min(xl[m], xl[rc[m]]);
    xr[m] = max(xr[m], xr[rc[m]]);
    yl[m] = min(yl[m], yl[rc[m]]);
    yr[m] = max(yr[m], yr[rc[m]]);
  }
  return m;
}
bool bound(const point &q, int o, long long d) {
  double ds = sqrt(d + 1.0);
  if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
    q.y < yl[o] - ds || q.y > yr[o] + ds)
    return false;
  return true;
}
long long dist(const point &a, const point &b) {
  return (a.x - b.x) * 1ll * (a.x - b.x) +
    (a.y - b.y) * 1ll * (a.y - b.y);
}
void dfs(
  const point &q, long long &d, int o, int dep = 0) {
  if (!bound(q, o, d)) return;
  long long cd = dist(p[o], q);
  if (cd != 0) d = min(d, cd);
  if ((dep & 1) && q.x < p[o].x ||
    !(dep & 1) && q.y < p[o].y) {
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
  } else {
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
  }
}
void init(const vector<point> &v) {
  for (int i = 0; i < v.size(); ++i) p[i] = v[i];
  root = build(0, v.size());
}
long long nearest(const point &q) {
  long long res = 1e18;
  dfs(q, res, root);
  return res;
}
} // namespace kdt
```

3.18 Range Chmin Chmax Add Range Sum [cd19b2]

```
#include <algorithm>
#include <iostream>
using namespace std;
typedef long long ll;

const int MAXC = 200005;
const ll INF = 1e18;

struct node {
  ll sum;
  ll mx, mxcnt, smx;
  ll mi, micnt, smi;
```

```
  ll lazymax, lazymin, lazyadd;
  node(ll k = 0)
    : sum(k), mx(k), mxcnt(1), smx(-INF), mi(k),
      micnt(1), smi(INF), lazymax(-INF), lazymin(INF),
      lazyadd(0) {}
  node operator+(const node &a) const {
    node rt;
    rt.sum = sum + a.sum;
    rt.mx = max(mx, a.mx);
    rt.mi = min(mi, a.mi);
    if (mx == a.mx) {
      rt.mxcnt = mxcnt + a.mxcnt;
      rt.smx = max(smx, a.smx);
    } else if (mx > a.mx) {
      rt.mxcnt = mxcnt;
      rt.smx = max(smx, a.mx);
    } else {
      rt.mxcnt = a.mxcnt;
      rt.smx = max(mx, a.smx);
    }
    if (mi == a.mi) {
      rt.micnt = micnt + a.micnt;
      rt.smi = min(smi, a.smi);
    } else if (mi < a.mi) {
      rt.micnt = micnt;
      rt.smi = min(smi, a.mi);
    } else {
      rt.micnt = a.micnt;
      rt.smi = min(mi, a.smi);
    }
    rt.lazymax = -INF;
    rt.lazymin = INF;
    rt.lazyadd = 0;
    return rt;
  }
} seg[MAXC << 2];

ll a[MAXC];

void give_tag_min(int rt, ll t) {
  if (t >= seg[rt].mx) return;
  seg[rt].lazymin = t;
  seg[rt].lazymax = min(seg[rt].lazymax, t);
  seg[rt].sum -= seg[rt].mxcnt * (seg[rt].mx - t);
  if (seg[rt].mx == seg[rt].smi) seg[rt].smi = t;
  if (seg[rt].mx == seg[rt].mi) seg[rt].mi = t;
  seg[rt].mx = t;
}

void give_tag_max(int rt, ll t) {
  if (t <= seg[rt].mi) return;
  seg[rt].lazymax = t;
  seg[rt].sum += seg[rt].micnt * (t - seg[rt].mi);
  if (seg[rt].mi == seg[rt].smx) seg[rt].smx = t;
  if (seg[rt].mi == seg[rt].mx) seg[rt].mx = t;
  seg[rt].mi = t;
}

void give_tag_add(int l, int r, int rt, ll t) {
  seg[rt].lazyadd += t;
  if (seg[rt].lazymax != -INF) seg[rt].lazymax += t;
  if (seg[rt].lazymin != INF) seg[rt].lazymin += t;
  seg[rt].mx += t;
  if (seg[rt].smx != -INF) seg[rt].smx += t;
  seg[rt].mi += t;
  if (seg[rt].smi != INF) seg[rt].smi += t;
  seg[rt].sum += (ll)(r - l + 1) * t;
}

void tag_down(int l, int r, int rt) {
  if (seg[rt].lazyadd != 0) {
    int mid = (l + r) >> 1;
    give_tag_add(l, mid, rt << 1, seg[rt].lazyadd);
    give_tag_add(
      mid + 1, r, rt << 1 | 1, seg[rt].lazyadd);
    seg[rt].lazyadd = 0;
  }
  if (seg[rt].lazymin != INF) {
    give_tag_min(rt << 1, seg[rt].lazymin);
    give_tag_min(rt << 1 | 1, seg[rt].lazymin);
    seg[rt].lazymin = INF;
  }
  if (seg[rt].lazymax != -INF) {
    give_tag_max(rt << 1, seg[rt].lazymax);
    give_tag_max(rt << 1 | 1, seg[rt].lazymax);
    seg[rt].lazymax = -INF;
  }
}

void build(int l, int r, int rt) {
  if (l == r) return seg[rt] = node(a[l]), void();
  int mid = (l + r) >> 1;
  build(l, mid, rt << 1);
  build(mid + 1, r, rt << 1 | 1);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

void modifymax(
  int L, int R, int l, int r, int rt, ll t) {
  if (L <= l && R >= r && t < seg[rt].smi)
    return give_tag_max(rt, t);
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (L <= mid) modifymax(L, R, l, mid, rt << 1, t);
  if (R > mid)
    modifymax(L, R, mid + 1, r, rt << 1 | 1, t);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

void modifymin(
  int L, int R, int l, int r, int rt, ll t) {
  if (L <= l && R >= r && t > seg[rt].smx)
    return give_tag_min(rt, t);
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (L <= mid) modifymin(L, R, l, mid, rt << 1, t);
  if (R > mid)
    modifymin(L, R, mid + 1, r, rt << 1 | 1, t);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

void modifyadd(
  int L, int R, int l, int r, int rt, ll t) {
  if (L <= l && R >= r)
    return give_tag_add(l, r, rt, t);
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (L <= mid) modifyadd(L, R, l, mid, rt << 1, t);
  if (R > mid)
    modifyadd(L, R, mid + 1, r, rt << 1 | 1, t);
  seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}

ll query(int L, int R, int l, int r, int rt) {
  if (L <= l && R >= r) return seg[rt].sum;
  if (l != r) tag_down(l, r, rt);
  int mid = (l + r) >> 1;
  if (R <= mid) return query(L, R, l, mid, rt << 1);
  if (L > mid)
    return query(L, R, mid + 1, r, rt << 1 | 1);
  return query(L, R, l, mid, rt << 1) +
    query(L, R, mid + 1, r, rt << 1 | 1);
}

int main() {
  ios::sync_with_stdio(0), cin.tie(0);
  int n, m;
  cin >> n >> m;
  for (int i = 1; i <= n; ++i) cin >> a[i];
  build(1, n, 1);
  while (m--) {
    int k, x, y;
    ll t;
    cin >> k >> x >> y, ++x;
    if (k == 0) cin >> t, modifymin(x, y, 1, n, 1, t);
    else if (k == 1)
      cin >> t, modifymax(x, y, 1, n, 1, t);
    else if (k == 2)
      cin >> t, modifyadd(x, y, 1, n, 1, t);
    else cout << query(x, y, 1, n, 1) << "\n";
  }
}
```

3.19 discrete trick [2062d6]

```
vector<int> val;
// build
sort(ALL(val)),
  val.resize(unique(ALL(val)) - val.begin());
// index of x
upper_bound(ALL(val), x) - val.begin();
// max idx <= x
upper_bound(ALL(val), x) - val.begin();
```

```
// max idx < x
lower_bound(ALL(val), x) - val.begin();
```

4 Flow Matching

4.1 Maximum Simple Graph Matching [390d20]

```
struct Matching { // 0-base
  queue<int> q;
  int n;
  vector<int> fa, s, vis, pre, match;
  vector<vector<int>> G;
  int Find(int u) {
    return u == fa[u] ? u : fa[u] = Find(fa[u]);
  }
  int LCA(int x, int y) {
    static int tk = 0;
    tk++;
    x = Find(x);
    y = Find(y);
    for (;; swap(x, y))
      if (x != n) {
        if (vis[x] == tk) return x;
        vis[x] = tk;
        x = Find(pre[match[x]]);
      }
  }
  void Blossom(int x, int y, int l) {
    for (; Find(x) != l; x = pre[y]) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      for (int z : {x, y})
        if (fa[z] == z) fa[z] = l;
    }
  }
  bool Bfs(int r) {
    iota(ALL(fa), 0);
    fill(ALL(s), -1);
    q = queue<int>();
    q.push(r);
    s[r] = 0;
    for (; !q.empty(); q.pop()) {
      for (int x = q.front(); int u : G[x])
        if (s[u] == -1) {
          if (pre[u] = x, s[u] = 1, match[u] == n) {
            for (int a = u, b = x, last; b != n;
                 a = last, b = pre[a])
              last = match[b], match[b] = a,
              match[a] = b;
            return true;
          }
          q.push(match[u]);
          s[match[u]] = 0;
        } else if (!s[u] && Find(u) != Find(x)) {
          int l = LCA(u, x);
          Blossom(x, u, l);
          Blossom(u, x, l);
        }
    }
    return false;
  }
  Matching(int _n)
    : n(_n), fa(n + 1), s(n + 1), vis(n + 1),
      pre(n + 1, n), match(n + 1, n), G(n) {}
  void add_edge(int u, int v) {
    G[u].pb(v), G[v].pb(u);
  }
  int solve() {
    int ans = 0;
    for (int x = 0; x < n; ++x)
      if (match[x] == n) ans += Bfs(x);
    return ans;
  } // match[x] == n means not matched
};
```

4.2 Kuhn Munkres [61bbd0]

```
struct KM { // 0-base, maximum matching
  ll w[N][N], hl[N], hr[N], slk[N];
  int fl[N], fr[N], pre[N], qu[N], ql, qr, n;
  bool vl[N], vr[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) fill_n(w[i], n, -INF);
  }
  void add_edge(int a, int b, ll wei) {
    w[a][b] = wei;
  }
  bool Check(int x) {
    if (vl[x] = 1, ~fl[x])
      return vr[qu[qr++] = fl[x]] = 1;
    while (~x) swap(x, fr[fl[x] = pre[x]]);
    return 0;
  }
  void bfs(int s) {
    fill_n(slk, n, INF), fill_n(vl, n, 0),
      fill_n(vr, n, 0);
    ql = qr = 0, qu[qr++] = s, vr[s] = 1;
    for (ll d;;) {
      while (ql < qr)
        for (int x = 0, y = qu[ql++]; x < n; ++x)
          if (!vl[x] &&
              slk[x] >= (d = hl[x] + hr[y] - w[x][y])) {
            if (pre[x] = y, d) slk[x] = d;
            else if (!Check(x)) return;
          }
      d = INF;
      for (int x = 0; x < n; ++x)
        if (!vl[x] && d > slk[x]) d = slk[x];
      for (int x = 0; x < n; ++x) {
        if (vl[x]) hl[x] += d;
        else slk[x] -= d;
        if (vr[x]) hr[x] -= d;
      }
      for (int x = 0; x < n; ++x)
        if (!vl[x] && !slk[x] && !Check(x)) return;
    }
  }
  ll solve() {
    fill_n(fl, n, -1), fill_n(fr, n, -1),
      fill_n(hr, n, 0);
    for (int i = 0; i < n; ++i)
      hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) bfs(i);
    ll res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
  }
};
```

4.3 Model

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x,y,l,u), connect $x \to y$ with capacity $u-l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X,Y)
  1. Redirect every edge: $y \to x$ if $(x,y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x,y,c), connect $x \to y$ with $(cost,cap) = (c,1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost,cap) = (-c,1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost,cap) = (0,d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost,cap) = (0,-d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C+K$
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u,v,w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$

6. T is a valid answer if the maximum flow $f < K|V|$

- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u,v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.
- Project selection problem
  1. If $p_v > 0$, create edge (s,v) with capacity p_v; otherwise, create edge (v,t) with capacity $-p_v$.
  2. Create edge (u,v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.
- Dual of minimum cost maximum flow
  1. Capacity c_{uv}, Flow f_{uv}, Cost w_{uv}, Required Flow difference for vertex b_u.
  2. If all w_{uv} are integers, then optimal solution can happen when all p_u are integers.

$$\min \sum_{uv} w_{uv} f_{uv} \qquad -f_{uv} \geq -c_{uv} \Leftrightarrow \qquad \min \sum_{u} b_u p_u + \sum_{uv} c_{uv} \max(0, p_v - p_u - w_{uv})$$

$$\sum_{v} f_{vu} - \sum_{v} f_{uv} = -b_u \qquad p_u \geq 0$$

4.4 MincostMaxflow dijkstra [94c520]

```cpp
struct MinCostMaxFlow { // 0-base
  struct Edge {
    ll from, to, cap, flow, cost, rev;
  } *past[N];
  vector<Edge> G[N];
  int inq[N], n, s, t;
  ll dis[N], up[N], pot[N];
  bool BellmanFord() {
    fill_n(dis, n, INF), fill_n(inq, n, 0);
    queue<int> q;
    auto relax = [&](int u, ll d, ll cap, Edge *e) {
      if (cap > 0 && dis[u] > d) {
        dis[u] = d, up[u] = cap, past[u] = e;
        if (!inq[u]) inq[u] = 1, q.push(u);
      }
    };
    relax(s, 0, INF, 0);
    while (!q.empty()) {
      int u = q.front();
      q.pop(), inq[u] = 0;
      for (auto &e : G[u]) {
        ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
        relax(
          e.to, d2, min(up[u], e.cap - e.flow), &e);
      }
    }
    return dis[t] != INF;
  }
  bool Dijkstra() {
    fill_n(dis, n, INF);
    priority_queue<pll, vector<pll>, greater<pll>> pq;
    auto relax = [&](int u, ll d, ll cap, Edge *e) {
      if (cap > 0 && dis[u] > d) {
        dis[u] = d, up[u] = cap, past[u] = e;
        pq.push(pll(d, u));
      }
    };
    relax(s, 0, INF, 0);
    while (!pq.empty()) {
      auto [d, u] = pq.top();
      pq.pop();
      if (dis[u] != d) continue;
      for (auto &e : G[u]) {
        ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
        relax(
          e.to, d2, min(up[u], e.cap - e.flow), &e);
      }
    }
    return dis[t] != INF;
  }
  void solve(int _s, int _t, ll &flow, ll &cost,
    bool neg = true) {
    s = _s, t = _t, flow = 0, cost = 0;
    if (neg) BellmanFord(), copy_n(dis, n, pot);
    for (; Dijkstra(); copy_n(dis, n, pot)) {
      for (int i = 0; i < n; ++i)
        dis[i] += pot[i] - pot[s];
      flow += up[t], cost += up[t] * dis[t];
      for (int i = t; past[i]; i = past[i]->from) {
        auto &e = *past[i];
        e.flow += up[t], G[e.to][e.rev].flow -= up[t];
      }
    }
  }
  void init(int _n) {
    n = _n, fill_n(pot, n, 0);
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void add_edge(ll a, ll b, ll cap, ll cost) {
    G[a].pb(Edge{a, b, cap, 0, cost, SZ(G[b])});
    G[b].pb(Edge{b, a, 0, 0, -cost, SZ(G[a]) - 1});
  }
};
```

4.5 isap [a2dc77]

```cpp
struct Maxflow {
  static const int MAXV = 20010;
  static const int INF = 1000000;
  struct Edge {
    int v, c, r;
    Edge(int _v, int _c, int _r)
      : v(_v), c(_c), r(_r) {}
  };
  int s, t;
  vector<Edge> G[MAXV * 2];
  int iter[MAXV * 2], d[MAXV * 2], gap[MAXV * 2], tot;
  void init(int x) {
    tot = x + 2;
    s = x + 1, t = x + 2;
    for (int i = 0; i <= tot; i++) {
      G[i].clear();
      iter[i] = d[i] = gap[i] = 0;
    }
  }
  void addEdge(int u, int v, int c) {
    G[u].push_back(Edge(v, c, SZ(G[v])));
    G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
  }
  int dfs(int p, int flow) {
    if (p == t) return flow;
    for (int &i = iter[p]; i < SZ(G[p]); i++) {
      Edge &e = G[p][i];
      if (e.c > 0 && d[p] == d[e.v] + 1) {
        int f = dfs(e.v, min(flow, e.c));
        if (f) {
          e.c -= f;
          G[e.v][e.r].c += f;
          return f;
        }
      }
    }
    if ((--gap[d[p]]) == 0) d[s] = tot;
    else {
      d[p]++;
      iter[p] = 0;
      ++gap[d[p]];
    }
    return 0;
  }
  int solve() {
    int res = 0;
    gap[0] = tot;
    for (res = 0; d[s] < tot; res += dfs(s, INF));
    return res;
  }
} flow;
```

4.6 Gomory Hu tree [62c88c]

```cpp
MaxFlow Dinic;
int g[MAXN];
void GomoryHu(int n) { // 0-base
  fill_n(g, n, 0);
  for (int i = 1; i < n; ++i) {
    Dinic.reset();
    add_edge(i, g[i], Dinic.maxflow(i, g[i]));
    for (int j = i + 1; j <= n; ++j)
      if (g[j] == g[i] && ~Dinic.dis[j]) g[j] = i;
  }
}
```

4.7 MincostMaxflow [0722e9]

```cpp
struct MinCostMaxFlow { // 0-base
  struct Edge {
    ll from, to, cap, flow, cost, rev;
  } *past[N];
```

```
  vector<Edge> G[N];
  int inq[N], n, s, t;
  ll dis[N], up[N], pot[N];
  bool BellmanFord() {
    fill_n(dis, n, INF), fill_n(inq, n, 0);
    queue<int> q;
    auto relax = [&](int u, ll d, ll cap, Edge *e) {
      if (cap > 0 && dis[u] > d) {
        dis[u] = d, up[u] = cap, past[u] = e;
        if (!inq[u]) inq[u] = 1, q.push(u);
      }
    };
    relax(s, 0, INF, 0);
    while (!q.empty()) {
      int u = q.front();
      q.pop(), inq[u] = 0;
      for (auto &e : G[u]) {
        ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
        relax(
          e.to, d2, min(up[u], e.cap - e.flow), &e);
      }
    }
    return dis[t] != INF;
  }
  void solve(int _s, int _t, ll &flow, ll &cost,
    bool neg = true) {
    s = _s, t = _t, flow = 0, cost = 0;
    if (neg) BellmanFord(), copy_n(dis, n, pot);
    for (; BellmanFord(); copy_n(dis, n, pot)) {
      for (int i = 0; i < n; ++i)
        dis[i] += pot[i] - pot[s];
      flow += up[t], cost += up[t] * dis[t];
      for (int i = t; past[i]; i = past[i]->from) {
        auto &e = *past[i];
        e.flow += up[t], G[e.to][e.rev].flow -= up[t];
      }
    }
  }
  void init(int _n) {
    n = _n, fill_n(pot, n, 0);
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void add_edge(ll a, ll b, ll cap, ll cost) {
    G[a].pb(Edge{a, b, cap, 0, cost, SZ(G[b])});
    G[b].pb(Edge{b, a, 0, 0, -cost, SZ(G[a]) - 1});
  }
};
```

4.8 SW-mincut [8e90f0]

```
struct SW { // global min cut, O(V^3)
#define REP for (int i = 0; i < n; ++i)
  static const int MXN = 514, INF = 2147483647;
  int vst[MXN], edge[MXN][MXN], wei[MXN];
  void init(int n) { REP fill_n(edge[i], n, 0); }
  void addEdge(int u, int v, int w) {
    edge[u][v] += w;
    edge[v][u] += w;
  }
  int search(int &s, int &t, int n) {
    fill_n(vst, n, 0), fill_n(wei, n, 0);
    s = t = -1;
    int mx, cur;
    for (int j = 0; j < n; ++j) {
      mx = -1, cur = 0;
      REP if (wei[i] > mx) cur = i, mx = wei[i];
      vst[cur] = 1, wei[cur] = -1;
      s = t;
      t = cur;
      REP if (!vst[i]) wei[i] += edge[cur][i];
    }
    return mx;
  }
  int solve(int n) {
    int res = INF;
    for (int x, y; n > 1; n--) {
      res = min(res, search(x, y, n));
      REP edge[i][x] = (edge[x][i] += edge[y][i]);
      REP {
        edge[y][i] = edge[n - 1][i];
        edge[i][y] = edge[i][n - 1];
      } // edge[y][y] = 0;
    }
    return res;
  }
} sw;
```

4.9 Maximum Weight Matching [a10467]

```
#define REP(i, l, r) for (int i = (l); i <= (r); ++i)
struct WeightGraph { // 1-based
  struct edge {
    int u, v, w;
  };
  int n, nx;
  vector<int> lab;
  vector<vector<edge>> g;
  vector<int> slk, match, st, pa, S, vis;
  vector<vector<int>> flo, flo_from;
  queue<int> q;
  WeightGraph(int n_)
    : n(n_), nx(n * 2), lab(nx + 1),
      g(nx + 1, vector<edge>(nx + 1)), slk(nx + 1),
      flo(nx + 1), flo_from(nx + 1, vector(n + 1, 0)) {
    match = st = pa = S = vis = slk;
    REP(u, 1, n) REP(v, 1, n) g[u][v] = {u, v, 0};
  }
  int E(edge e) {
    return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
  }
  void update_slk(int u, int x, int &s) {
    if (!s || E(g[u][x]) < E(g[s][x])) s = u;
  }
  void set_slk(int x) {
    slk[x] = 0;
    REP(u, 1, n)
    if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
      update_slk(u, x, slk[x]);
  }
  void q_push(int x) {
    if (x <= n) q.push(x);
    else
      for (int y : flo[x]) q_push(y);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n)
      for (int y : flo[x]) set_st(y, b);
  }
  vector<int> split_flo(auto &f, int xr) {
    auto it = find(ALL(f), xr);
    if (auto pr = it - f.begin(); pr % 2 == 1)
      reverse(1 + ALL(f)), it = f.end() - pr;
    auto res = vector(f.begin(), it);
    return f.erase(f.begin(), it), res;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    int xr = flo_from[u][g[u][v].u];
    auto &f = flo[u], z = split_flo(f, xr);
    REP(i, 0, SZ(z) - 1) set_match(z[i], z[i ^ 1]);
    set_match(xr, v);
    f.insert(f.end(), ALL(z));
  }
  void augment(int u, int v) {
    for (;;) {
      int xnv = st[match[u]];
      set_match(u, v);
      if (!xnv) return;
      set_match(v = xnv, u = st[pa[xnv]]);
    }
  }
  int lca(int u, int v) {
    static int t = 0;
    ++t;
    for (++t; u || v; swap(u, v))
      if (u) {
        if (vis[u] == t) return u;
        vis[u] = t, u = st[match[u]];
        if (u) u = st[pa[u]];
      }
    return 0;
  }
  void add_blossom(int u, int o, int v) {
    int b = find(n + 1 + ALL(st), 0) - begin(st);
    lab[b] = 0, S[b] = 0, match[b] = match[o];
    vector<int> f = {o};
    for (int t : {u, v}) {
      reverse(1 + ALL(f));
      for (int x = t, y; x != o; x = st[pa[y]])
        f.pb(x), f.pb(y = st[match[x]]), q_push(y);
    }
```

```
    flo[b] = f;
    set_st(b, b);
    REP(x, 1, nx) g[b][x].w = g[x][b].w = 0;
    fill(ALL(flo_from[b]), 0);
    for (int xs : flo[b]) {
      REP(x, 1, nx)
      if (g[b][x].w == 0 || E(g[xs][x]) < E(g[b][x]))
        g[b][x] = g[xs][x], g[x][b] = g[x][xs];
      REP(x, 1, n)
      if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slk(b);
  }
  void expand_blossom(int b) {
    for (int x : flo[b]) set_st(x, x);
    int xr = flo_from[b][g[b][pa[b]].u], xs = -1;
    for (int x : split_flo(flo[b], xr)) {
      if (xs == -1) {
        xs = x;
        continue;
      }
      pa[xs] = g[x][xs].u, S[xs] = 1, S[x] = 0;
      slk[xs] = 0, set_slk(x), q_push(x), xs = -1;
    }
    for (int x : flo[b])
      if (x == xr) S[x] = 1, pa[x] = pa[b];
      else S[x] = -1, set_slk(x);
    st[b] = 0;
  }
  bool on_found_edge(const edge &e) {
    if (int u = st[e.u], v = st[e.v]; S[v] == -1) {
      int nu = st[match[v]];
      pa[v] = e.u;
      S[v] = 1;
      slk[v] = slk[nu] = S[nu] = 0;
      q_push(nu);
    } else if (S[v] == 0) {
      if (int o = lca(u, v)) add_blossom(u, o, v);
      else return augment(u, v), augment(v, u), true;
    }
    return false;
  }
  bool matching() {
    fill(ALL(S), -1), fill(ALL(slk), 0);
    q = queue<int>();
    REP(x, 1, nx)
    if (st[x] == x && !match[x]) pa[x] = S[x] = 0,
                                 q_push(x);
    if (q.empty()) return false;
    for (;;) {
      while (SZ(q)) {
        int u = q.front();
        q.pop();
        if (S[st[u]] == 1) continue;
        REP(v, 1, n)
        if (g[u][v].w > 0 && st[u] != st[v]) {
          if (E(g[u][v]) != 0)
            update_slk(u, st[v], slk[st[v]]);
          else if (on_found_edge(g[u][v])) return true;
        }
      }
      int d = INF;
      REP(b, n + 1, nx)
      if (st[b] == b && S[b] == 1) d =
        min(d, lab[b] / 2);
      REP(x, 1, nx)
      if (int s = slk[x]; st[x] == x && s && S[x] <= 0)
        d = min(d, E(g[s][x]) / (S[x] + 2));
      REP(u, 1, n)
      if (S[st[u]] == 1) lab[u] += d;
      else if (S[st[u]] == 0) {
        if (lab[u] <= d) return false;
        lab[u] -= d;
      }
      REP(b, n + 1, nx)
      if (st[b] == b && S[b] >= 0) lab[b] +=
        d * (2 - 4 * S[b]);
      REP(x, 1, nx)
      if (int s = slk[x]; st[x] == x && s &&
          st[s] != x && E(g[s][x]) == 0)
        if (on_found_edge(g[s][x])) return true;
      REP(b, n + 1, nx)
      if (st[b] == b && S[b] == 1 && lab[b] == 0)
        expand_blossom(b);
    }
    return false;
  }
  pair<ll, int> solve() {
    fill(ALL(match), 0);
    REP(u, 0, n) st[u] = u, flo[u].clear();
    int w_max = 0;
    REP(u, 1, n) REP(v, 1, n) {
      flo_from[u][v] = (u == v ? u : 0);
      w_max = max(w_max, g[u][v].w);
    }
    fill(ALL(lab), w_max);
    int n_matches = 0;
    ll tot_weight = 0;
    while (matching()) ++n_matches;
    REP(u, 1, n)
    if (match[u] && match[u] < u) tot_weight +=
      g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
  }
  void add_edge(int u, int v, int w) {
    g[u][v].w = g[v][u].w = w;
  }
};
```

4.10 Minimum Weight Matching wrong [f27d66]

```
struct Graph { // 0-base (Perfect Match), n is even
  int n, match[N], onstk[N], stk[N], tp;
  ll edge[N][N], dis[N];
  void init(int _n) {
    n = _n, tp = 0;
    for (int i = 0; i < n; ++i) fill_n(edge[i], n, 0);
  }
  void add_edge(int u, int v, ll w) {
    edge[u][v] = edge[v][u] = w;
  }
  bool SPFA(int u) {
    stk[tp++] = u, onstk[u] = 1;
    for (int v = 0; v < n; ++v)
      if (!onstk[v] && match[u] != v) {
        int m = match[v];
        if (dis[m] >
          dis[u] - edge[v][m] + edge[u][v]) {
          dis[m] = dis[u] - edge[v][m] + edge[u][v];
          onstk[v] = 1, stk[tp++] = v;
          if (onstk[m] || SPFA(m)) return 1;
          --tp, onstk[v] = 0;
        }
      }
    onstk[u] = 0, --tp;
    return 0;
  }
  ll solve() { // find a match
    for (int i = 0; i < n; ++i) match[i] = i ^ 1;
    while (1) {
      int found = 0;
      fill_n(dis, n, 0);
      fill_n(onstk, n, 0);
      for (int i = 0; i < n; ++i)
        if (tp = 0, !onstk[i] && SPFA(i))
          for (found = 1; tp >= 2;) {
            int u = stk[--tp];
            int v = stk[--tp];
            match[u] = v, match[v] = u;
          }
      if (!found) break;
    }
    ll ret = 0;
    for (int i = 0; i < n; ++i)
      ret += edge[i][match[i]];
    return ret >> 1;
  }
};
```

4.11 Bipartite Matching [623c76]

```
struct Bipartite_Matching { // 0-base
  int mp[N], mq[N], dis[N + 1], cur[N], l, r;
  vector<int> G[N + 1];
  bool dfs(int u) {
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      int e = G[u][i];
      if (mq[e] == l ||
        (dis[mq[e]] == dis[u] + 1 && dfs(mq[e])))
        return mp[mq[e] = u] = e, 1;
    }
    return dis[u] = -1, 0;
  }
```

```
  bool bfs() {
    queue<int> q;
    fill_n(dis, l + 1, -1);
    for (int i = 0; i < l; ++i)
      if (!~mp[i]) q.push(i), dis[i] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int e : G[u])
        if (!~dis[mq[e]])
          q.push(mq[e]), dis[mq[e]] = dis[u] + 1;
    }
    return dis[l] != -1;
  }
  int matching() {
    int res = 0;
    fill_n(mp, l, -1), fill_n(mq, r, l);
    while (bfs()) {
      fill_n(cur, l, 0);
      for (int i = 0; i < l; ++i)
        res += (!~mp[i] && dfs(i));
    }
    return res; // (i, mp[i] != -1)
  }
  void add_edge(int s, int t) { G[s].pb(t); }
  void init(int _l, int _r) {
    l = _l, r = _r;
    for (int i = 0; i <= l; ++i) G[i].clear();
  }
};
```

4.12 BoundedFlow [e8670b]

```
struct BoundedFlow { // 0-base
  struct edge {
    int to, cap, flow, rev;
  };
  vector<edge> G[N];
  int n, s, t, dis[N], cur[N], cnt[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n + 2; ++i)
      G[i].clear(), cnt[i] = 0;
  }
  void add_edge(int u, int v, int lcap, int rcap) {
    cnt[u] -= lcap, cnt[v] += lcap;
    G[u].pb(edge{v, rcap, lcap, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  void add_edge(int u, int v, int cap) {
    G[u].pb(edge{v, cap, 0, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  int dfs(int u, int cap) {
    if (u == t || !cap) return cap;
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
        int df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df, G[e.to][e.rev].flow -= df;
          return df;
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n + 3, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (edge &e : G[u])
        if (!~dis[e.to] && e.flow != e.cap)
          q.push(e.to), dis[e.to] = dis[u] + 1;
    }
    return dis[t] != -1;
  }
  int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
      fill_n(cur, n + 3, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
  bool solve() {
    int sum = 0;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        add_edge(n + 1, i, cnt[i]), sum += cnt[i];
      else if (cnt[i] < 0) add_edge(i, n + 2, -cnt[i]);
    if (sum != maxflow(n + 1, n + 2)) sum = -1;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        G[n + 1].pop_back(), G[i].pop_back();
      else if (cnt[i] < 0)
        G[i].pop_back(), G[n + 2].pop_back();
    return sum != -1;
  }
  int solve(int _s, int _t) {
    add_edge(_t, _s, INF);
    if (!solve()) return -1; // invalid flow
    int x = G[_t].back().flow;
    return G[_t].pop_back(), G[_s].pop_back(), x;
  }
};
```

4.13 Dinic [ba0999]

```
struct MaxFlow { // 0-base
  struct edge {
    int to, cap, flow, rev;
  };
  vector<edge> G[MAXN];
  int s, t, dis[MAXN], cur[MAXN], n;
  int dfs(int u, int cap) {
    if (u == t || !cap) return cap;
    for (int &i = cur[u]; i < (int)G[u].size(); ++i) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
        int df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df;
          G[e.to][e.rev].flow -= df;
          return df;
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
      int tmp = q.front();
      q.pop();
      for (auto &u : G[tmp])
        if (!~dis[u.to] && u.flow != u.cap) {
          q.push(u.to);
          dis[u.to] = dis[tmp] + 1;
        }
    }
    return dis[t] != -1;
  }
  int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
      fill_n(cur, n, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void reset() {
    for (int i = 0; i < n; ++i)
      for (auto &j : G[i]) j.flow = 0;
  }
  void add_edge(int u, int v, int cap) {
    G[u].pb(edge{v, cap, 0, (int)G[v].size()});
    G[v].pb(edge{u, 0, 0, (int)G[u].size() - 1});
  }
};
```

4.14 MinCostCirculation [86e6a8]

```cpp
struct MinCostCirculation { // 0-base
  struct Edge {
    ll from, to, cap, fcap, flow, cost, rev;
  } *past[N];
  vector<Edge> G[N];
  ll dis[N], inq[N], n;
  void BellmanFord(int s) {
    fill_n(dis, n, INF), fill_n(inq, n, 0);
    queue<int> q;
    auto relax = [&](int u, ll d, Edge *e) {
      if (dis[u] > d) {
        dis[u] = d, past[u] = e;
        if (!inq[u]) inq[u] = 1, q.push(u);
      }
    };
    relax(s, 0, 0);
    while (!q.empty()) {
      int u = q.front();
      q.pop(), inq[u] = 0;
      for (auto &e : G[u])
        if (e.cap > e.flow)
          relax(e.to, dis[u] + e.cost, &e);
    }
  }
  void try_edge(Edge &cur) {
    if (cur.cap > cur.flow) return ++cur.cap, void();
    BellmanFord(cur.to);
    if (dis[cur.from] + cur.cost < 0) {
      ++cur.flow, --G[cur.to][cur.rev].flow;
      for (int i = cur.from; past[i];
           i = past[i]->from) {
        auto &e = *past[i];
        ++e.flow, --G[e.to][e.rev].flow;
      }
    }
    ++cur.cap;
  }
  void solve(int mxlg) {
    for (int b = mxlg; b >= 0; --b) {
      for (int i = 0; i < n; ++i)
        for (auto &e : G[i]) e.cap *= 2, e.flow *= 2;
      for (int i = 0; i < n; ++i)
        for (auto &e : G[i])
          if (e.fcap >> b & 1) try_edge(e);
    }
  }
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void add_edge(ll a, ll b, ll cap, ll cost) {
    G[a].pb(Edge{
      a, b, 0, cap, 0, cost, SZ(G[b]) + (a == b)});
    G[b].pb(Edge{b, a, 0, 0, 0, -cost, SZ(G[a]) - 1});
  }
} mcmf; // O(VE * ElogC)
```

5 String

5.1 Smallest Rotation [d69462]

```cpp
string mcp(string s) {
  int n = SZ(s), i = 0, j = 1;
  s += s;
  while (i < n && j < n) {
    int k = 0;
    while (k < n && s[i + k] == s[j + k]) ++k;
    if (s[i + k] <= s[j + k]) j += k + 1;
    else i += k + 1;
    if (i == j) ++j;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```

5.2 Manacher [11ebce]

```cpp
int z[MAXN]; // 0-base
/* center i: radius z[i * 2 + 1] / 2
   center i, i + 1: radius z[i * 2 + 2] / 2
   both aba, abba have radius 2 */
void Manacher(string tmp) {
  string s = "%";
  int l = 0, r = 0;
  for (char c : tmp) s.pb(c), s.pb('%');
  for (int i = 0; i < SZ(s); ++i) {
    z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
    while (i - z[i] >= 0 && i + z[i] < SZ(s) &&
      s[i + z[i]] == s[i - z[i]])
      ++z[i];
    if (z[i] + i > r) r = z[i] + i, l = i;
  }
}
```

5.3 De Bruijn sequence [151f80]

```cpp
constexpr int MAXC = 10, MAXN = 1e5 + 10;
struct DBSeq {
  int C, N, K, L, buf[MAXC * MAXN]; // K <= C^N
  void dfs(int *out, int t, int p, int &ptr) {
    if (ptr >= L) return;
    if (t > N) {
      if (N % p) return;
      for (int i = 1; i <= p && ptr < L; ++i)
        out[ptr++] = buf[i];
    } else {
      buf[t] = buf[t - p], dfs(out, t + 1, p, ptr);
      for (int j = buf[t - p] + 1; j < C; ++j)
        buf[t] = j, dfs(out, t + 1, t, ptr);
    }
  }
  void solve(int _c, int _n, int _k, int *out) {
    int p = 0;
    C = _c, N = _n, K = _k, L = N + K - 1;
    dfs(out, 1, 1, p);
    if (p < L) fill(out + p, out + L, 0);
  }
} dbs;
```

5.4 SAM [4d0baa]

```cpp
const int MAXM = 1000010;
struct SAM {
  int tot, root, lst, mom[MAXM], mx[MAXM];
  int nxt[MAXM][33], cnt[MAXM], in[MAXM];
  int newNode() {
    int res = ++tot;
    fill(nxt[res], nxt[res] + 33, 0);
    mom[res] = mx[res] = cnt[res] = in[res] = 0;
    return res;
  }
  void init() {
    tot = 0;
    root = newNode();
    mom[root] = 0, mx[root] = 0;
    lst = root;
  }
  void push(int c) {
    int p = lst;
    int np = newNode();
    mx[np] = mx[p] + 1;
    for (; p && nxt[p][c] == 0; p = mom[p])
      nxt[p][c] = np;
    if (p == 0) mom[np] = root;
    else {
      int q = nxt[p][c];
      if (mx[p] + 1 == mx[q]) mom[np] = q;
      else {
        int nq = newNode();
        mx[nq] = mx[p] + 1;
        for (int i = 0; i < 33; i++)
          nxt[nq][i] = nxt[q][i];
        mom[nq] = mom[q];
        mom[q] = nq;
        mom[np] = nq;
        for (; p && nxt[p][c] == q; p = mom[p])
          nxt[p][c] = nq;
      }
    }
    lst = np, cnt[np] = 1;
  }
  void push(char *str) {
    for (int i = 0; str[i]; i++)
      push(str[i] - 'a' + 1);
  }
  void count() {
    for (int i = 1; i <= tot; ++i) ++in[mom[i]];
    queue<int> q;
    for (int i = 1; i <= tot; ++i)
      if (!in[i]) q.push(i);
    while (!q.empty()) {
      int u = q.front();
```

```
      q.pop();
      cnt[mom[u]] += cnt[u];
      if (!--in[mom[u]]) q.push(mom[u]);
    }
  }
} sam;
```

5.5 Aho-Corasick Automatan [8c56e8]

```
struct AC_Automatan {
  int nx[len][sigma], fl[len], cnt[len], ord[len], top;
  int rnx[len][sigma]; // node actually be reached
  int newnode() {
    fill_n(nx[top], sigma, -1);
    return top++;
  }
  void init() { top = 1, newnode(); }
  int input(string &s) {
    int X = 1;
    for (char c : s) {
      if (!~nx[X][c - 'A']) nx[X][c - 'A'] = newnode();
      X = nx[X][c - 'A'];
    }
    return X; // return the end node of string
  }
  void make_fl() {
    queue<int> q;
    q.push(1), fl[1] = 0;
    for (int t = 0; !q.empty();) {
      int R = q.front();
      q.pop(), ord[t++] = R;
      for (int i = 0; i < sigma; ++i)
        if (~nx[R][i]) {
          int X = rnx[R][i] = nx[R][i], Z = fl[R];
          for (; Z && !~nx[Z][i];) Z = fl[Z];
          fl[X] = Z ? nx[Z][i] : 1, q.push(X);
        } else rnx[R][i] = R > 1 ? rnx[fl[R]][i] : 1;
    }
  }
  void solve() {
    for (int i = top - 2; i > 0; --i)
      cnt[fl[ord[i]]] += cnt[ord[i]];
  }
} ac;
```

5.6 SAIS-old [ea9200]

```
class SAIS {
public:
  int *SA, *H;
  // zero based, string content MUST > 0
  // result height H[i] is LCP(SA[i - 1], SA[i])
  // string, length, |sigma|
  void build(int *s, int n, int m = 128) {
    copy_n(s, n, _s);
    _h[0] = _s[n++] = 0;
    sais(_s, _sa, _p, _q, _t, _c, n, m);
    mkhei(n);
    SA = _sa + 1;
    H = _h + 1;
  }

private:
  bool _t[N * 2];
  int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2],
    r[N], _sa[N * 2], _h[N];
  void mkhei(int n) {
    for (int i = 0; i < n; i++) r[_sa[i]] = i;
    for (int i = 0; i < n; i++)
      if (r[i]) {
        int ans = i > 0 ? max(_h[r[i - 1]] - 1, 0) : 0;
        while (_s[i + ans] == _s[_sa[r[i] - 1] + ans])
          ans++;
        _h[r[i]] = ans;
      }
  }
  void sais(int *s, int *sa, int *p, int *q, bool *t,
    int *c, int n, int z) {
    bool uniq = t[n - 1] = 1, neq;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
        lst = -1;

#define MAGIC(XD)                                \
  fill_n(sa, n, 0);                              \
  copy_n(c, z, x);                               \
  XD;                                            \
  copy_n(c, z - 1, x + 1);                       \
  for (int i = 0; i < n; i++)                    \
    if (sa[i] && !t[sa[i] - 1])                  \
      sa[x[s[sa[i] - 1]]++] = sa[i] - 1;         \
  copy_n(c, z, x);                               \
  for (int i = n - 1; i >= 0; i--)               \
    if (sa[i] && t[sa[i] - 1])                   \
      sa[--x[s[sa[i] - 1]]] = sa[i] - 1;

    fill_n(c, z, 0);
    for (int i = 0; i < n; i++) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) {
      for (int i = 0; i < n; i++) sa[--c[s[i]]] = i;
      return;
    }
    for (int i = n - 2; i >= 0; i--)
      t[i] = (s[i] == s[i + 1] ? t[i + 1]
                               : s[i] < s[i + 1]);
    MAGIC(for (int i = 1; i <= n - 1;
               i++) if (t[i] && !t[i - 1])
              sa[--x[s[i]]] = p[q[i] = nn++] = i);
    for (int i = 0; i < n; i++)
      if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
        neq = (lst < 0) ||
          !equal(s + lst,
            s + lst + p[q[sa[i]] + 1] - sa[i],
            s + sa[i]);
        ns[q[lst = sa[i]]] = nmxz += neq;
      }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
      nmxz + 1);
    MAGIC(for (int i = nn - 1; i >= 0; i--)
              sa[--x[s[p[nsa[i]]]]] = p[nsa[i]]);
  }
} sa;
```

5.7 Z-value [2e5c4c]

```
int z[MAXn];
void make_z(const string &s) {
  int l = 0, r = 0;
  for (int i = 1; i < SZ(s); ++i) {
    for (z[i] = max(0, min(r - i + 1, z[i - l]));
         i + z[i] < SZ(s) && s[i + z[i]] == s[z[i]];
         ++z[i]);
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
}
```

5.8 exSAM [0b980b]

```
struct exSAM {
  int len[N * 2], link[N * 2]; // maxlength, suflink
  int next[N * 2][CNUM], tot; // [0, tot), root = 0
  int lenSorted[N * 2]; // topo. order
  int cnt[N * 2]; // occurence
  int newnode() {
    fill_n(next[tot], CNUM, 0);
    len[tot] = cnt[tot] = link[tot] = 0;
    return tot++;
  }
  void init() { tot = 0, newnode(), link[0] = -1; }
  int insertSAM(int last, int c) {
    int cur = next[last][c];
    len[cur] = len[last] + 1;
    int p = link[last];
    while (p != -1 && !next[p][c])
      next[p][c] = cur, p = link[p];
    if (p == -1) return link[cur] = 0, cur;
    int q = next[p][c];
    if (len[p] + 1 == len[q])
      return link[cur] = q, cur;
    int clone = newnode();
    for (int i = 0; i < CNUM; ++i)
      next[clone][i] =
        len[next[q][i]] ? next[q][i] : 0;
    len[clone] = len[p] + 1;
    while (p != -1 && next[p][c] == q)
      next[p][c] = clone, p = link[p];
    link[link[cur] = clone] = link[q];
    link[q] = clone;
    return cur;
  }
  void insert(const string &s) {
    int cur = 0;
    for (auto ch : s) {
      int &nxt = next[cur][int(ch - 'a')];
```

```
      if (!nxt) nxt = newnode();
      cnt[cur = nxt] += 1;
    }
  }
  void build() {
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
      int cur = q.front();
      q.pop();
      for (int i = 0; i < CNUM; ++i)
        if (next[cur][i]) q.push(insertSAM(cur, i));
    }
    vector<int> lc(tot);
    for (int i = 1; i < tot; ++i) ++lc[len[i]];
    partial_sum(ALL(lc), lc.begin());
    for (int i = 1; i < tot; ++i)
      lenSorted[--lc[len[i]]] = i;
  }
  void solve() {
    for (int i = tot - 2; i >= 0; --i)
      cnt[link[lenSorted[i]]] += cnt[lenSorted[i]];
  }
};
```

5.9 SAIS [8306e4]

```
namespace sfx {
bool _t[N * 2];
int SA[N * 2], H[N], RA[N];
int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2];
// zero based, string content MUST > 0
// SA[i]: SA[i]-th suffix is the i-th lexigraphically
// smallest suffix. H[i]: longest common prefix of
// suffix SA[i] and suffix SA[i - 1].
void pre(int *sa, int *c, int n, int z) {
  fill_n(sa, n, 0), copy_n(c, z, x);
}
void induce(
  int *sa, int *c, int *s, bool *t, int n, int z) {
  copy_n(c, z - 1, x + 1);
  for (int i = 0; i < n; ++i)
    if (sa[i] && !t[sa[i] - 1])
      sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
  copy_n(c, z, x);
  for (int i = n - 1; i >= 0; --i)
    if (sa[i] && t[sa[i] - 1])
      sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
}
void sais(int *s, int *sa, int *p, int *q, bool *t,
  int *c, int n, int z) {
  bool uniq = t[n - 1] = true;
  int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
      last = -1;
  fill_n(c, z, 0);
  for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
  partial_sum(c, c + z, c);
  if (uniq) {
    for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
    return;
  }
  for (int i = n - 2; i >= 0; --i)
    t[i] =
      (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
  pre(sa, c, n, z);
  for (int i = 1; i <= n - 1; ++i)
    if (t[i] && !t[i - 1])
      sa[--x[s[i]]] = p[q[i] = nn++] = i;
  induce(sa, c, s, t, n, z);
  for (int i = 0; i < n; ++i)
    if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
      bool neq = last < 0 ||
        !equal(
          s + sa[i], s + p[q[sa[i]] + 1], s + last);
      ns[q[last = sa[i]]] = nmxz += neq;
    }
  sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
    nmxz + 1);
  pre(sa, c, n, z);
  for (int i = nn - 1; i >= 0; --i)
    sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
  induce(sa, c, s, t, n, z);
}
void mkhei(int n) {
  for (int i = 0, j = 0; i < n; ++i) {
    if (RA[i])
      for (; _s[i + j] == _s[SA[RA[i] - 1] + j]; ++j);
    H[RA[i]] = j, j = max(0, j - 1);
  }
}
void build(int *s, int n) {
  copy_n(s, n, _s), _s[n] = 0;
  sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
  copy_n(SA + 1, n, SA);
  for (int i = 0; i < n; ++i) RA[SA[i]] = i;
  mkhei(n);
}
} // namespace sfx
```

5.10 SAIS-C++20 [b8cdc4]

```
auto sais(const auto &s) {
  const int n = SZ(s), z = ranges::max(s) + 1;
  if (n == 1) return vector{0};
  vector<int> c(z);
  for (int x : s) ++c[x];
  partial_sum(ALL(c), begin(c));
  vector<int> sa(n);
  auto I = views::iota(0, n);
  vector<bool> t(n, true);
  for (int i = n - 2; i >= 0; --i)
    t[i] =
      (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
  auto is_lms = views::filter(
    [&t](int x) { return x && t[x] && !t[x - 1]; });
  auto induce = [&] {
    for (auto x = c; int y : sa)
      if (y--)
        if (!t[y]) sa[x[s[y] - 1]++] = y;
    for (auto x = c; int y : sa | views::reverse)
      if (y--)
        if (t[y]) sa[--x[s[y]]] = y;
  };
  vector<int> lms, q(n);
  lms.reserve(n);
  for (auto x = c; int i : I | is_lms)
    q[i] = SZ(lms), lms.pb(sa[--x[s[i]]] = i);
  induce();
  vector<int> ns(SZ(lms));
  for (int j = -1, nz = 0; int i : sa | is_lms) {
    if (j >= 0) {
      int len = min({n - i, n - j, lms[q[i] + 1] - i});
      ns[q[i]] = nz += lexicographical_compare(
        begin(s) + j, begin(s) + j + len, begin(s) + i,
        begin(s) + i + len);
    }
    j = i;
  }
  fill(ALL(sa), 0);
  auto nsa = sais(ns);
  for (auto x = c; int y : nsa | views::reverse)
    y = lms[y], sa[--x[s[y]]] = y;
  return induce(), sa;
}
// sa[i]: sa[i]-th suffix is the i-th lexicographically
// smallest suffix. hi[i]: LCP of suffix sa[i] and
// suffix sa[i - 1].
struct Suffix {
  int n;
  vector<int> sa, hi, ra;
  Suffix(const auto &_s, int _n)
    : n(_n), hi(n), ra(n) {
    vector<int> s(n + 1); // s[n] = 0;
    copy_n(_s, n, begin(s)); // _s shouldn't contain 0
    sa = sais(s);
    sa.erase(sa.begin());
    for (int i = 0; i < n; ++i) ra[sa[i]] = i;
    for (int i = 0, h = 0; i < n; ++i) {
      if (!ra[i]) {
        h = 0;
        continue;
      }
      for (int j = sa[ra[i] - 1];
           max(i, j) + h < n && s[i + h] == s[j + h];)
        ++h;
      hi[ra[i]] = h ? h-- : 0;
    }
  }
};
```

5.11 PalTree [9bd3fb]

```
struct palindromic_tree {
  struct node {
```

```
    int next[26], fail, len;
    int cnt, num; // cnt: appear times, num: number of
                  // pal. suf.
    node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
      for (int i = 0; i < 26; ++i) next[i] = 0;
    }
  };
  vector<node> St;
  vector<char> s;
  int last, n;
  palindromic_tree() : St(2), last(1), n(0) {
    St[0].fail = 1, St[1].len = -1, s.pb(-1);
  }
  inline void clear() {
    St.clear(), s.clear(), last = 1, n = 0;
    St.pb(0), St.pb(-1);
    St[0].fail = 1, s.pb(-1);
  }
  inline int get_fail(int x) {
    while (s[n - St[x].len - 1] != s[n])
      x = St[x].fail;
    return x;
  }
  inline void add(int c) {
    s.push_back(c -= 'a'), ++n;
    int cur = get_fail(last);
    if (!St[cur].next[c]) {
      int now = SZ(St);
      St.pb(St[cur].len + 2);
      St[now].fail =
        St[get_fail(St[cur].fail)].next[c];
      St[cur].next[c] = now;
      St[now].num = St[St[now].fail].num + 1;
    }
    last = St[cur].next[c], ++St[last].cnt;
  }
  inline void count() { // counting cnt
    auto i = St.rbegin();
    for (; i != St.rend(); ++i) {
      St[i->fail].cnt += i->cnt;
    }
  }
  inline int size() { // The number of diff. pal.
    return SZ(St) - 2;
  }
};
```

5.12 MainLorentz [2981c4]

```
vector<pair<int, int>> rep[kN]; // 0-base [l, r]
void main_lorentz(const string &s, int sft = 0) {
  const int n = s.size();
  if (n == 1) return;
  const int nu = n / 2, nv = n - nu;
  const string u = s.substr(0, nu), v = s.substr(nu),
               ru(u.rbegin(), u.rend()),
               rv(v.rbegin(), v.rend());
  main_lorentz(u, sft), main_lorentz(v, sft + nu);
  const auto z1 = Zalgo(ru), z2 = Zalgo(v + '#' + u),
             z3 = Zalgo(ru + '#' + rv), z4 = Zalgo(v);
  auto get_z = [](const vector<int> &z, int i) {
    return (0 <= i and i < (int)z.size()) ? z[i] : 0;
  };
  auto add_rep = [&](bool left, int c, int l, int k1,
                     int k2) {
    const int L = max(1, l - k2),
              R = min(l - left, k1);
    if (L > R) return;
    if (left)
      rep[l].emplace_back(sft + c - R, sft + c - L);
    else
      rep[l].emplace_back(
        sft + c - R - l + 1, sft + c - L - l + 1);
  };
  for (int cntr = 0; cntr < n; cntr++) {
    int l, k1, k2;
    if (cntr < nu) {
      l = nu - cntr;
      k1 = get_z(z1, nu - cntr);
      k2 = get_z(z2, nv + 1 + cntr);
    } else {
      l = cntr - nu + 1;
      k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
      k2 = get_z(z4, (cntr - nu) + 1);
    }
    if (k1 + k2 >= l)
      add_rep(cntr < nu, cntr, l, k1, k2);
  }
} // p \in [l, r] => s[p, p + i) = s[p + i, p + 2i)
```

5.13 KMP [32f229]

```
int F[MAXN];
vector<int> match(string A, string B) {
  vector<int> ans;
  F[0] = -1, F[1] = 0;
  for (int i = 1, j = 0; i < SZ(B); F[++i] = ++j) {
    if (B[i] == B[j]) F[i] = F[j]; // optimize
    while (j != -1 && B[i] != B[j]) j = F[j];
  }
  for (int i = 0, j = 0; i < SZ(A); ++i) {
    while (j != -1 && A[i] != B[j]) j = F[j];
    if (++j == SZ(B)) ans.pb(i + 1 - j), j = F[j];
  }
  return ans;
}
```

5.14 Suffix Array [b981d5]

```
struct suffix_array {
  int box[MAXN], tp[MAXN], m;
  bool not_equ(int a, int b, int k, int n) {
    return ra[a] != ra[b] || a + k >= n ||
      b + k >= n || ra[a + k] != ra[b + k];
  }
  void radix(int *key, int *it, int *ot, int n) {
    fill_n(box, m, 0);
    for (int i = 0; i < n; ++i) ++box[key[i]];
    partial_sum(box, box + m, box);
    for (int i = n - 1; i >= 0; --i)
      ot[--box[key[it[i]]]] = it[i];
  }
  void make_sa(const string &s, int n) {
    int k = 1;
    for (int i = 0; i < n; ++i) ra[i] = s[i];
    do {
      iota(tp, tp + k, n - k), iota(sa + k, sa + n, 0);
      radix(ra + k, sa + k, tp + k, n - k);
      radix(ra, tp, sa, n);
      tp[sa[0]] = 0, m = 1;
      for (int i = 1; i < n; ++i) {
        m += not_equ(sa[i], sa[i - 1], k, n);
        tp[sa[i]] = m - 1;
      }
      copy_n(tp, n, ra);
      k *= 2;
    } while (k < n && m != n);
  }
  void make_he(const string &s, int n) {
    for (int j = 0, k = 0; j < n; ++j) {
      if (ra[j])
        for (; s[j + k] == s[sa[ra[j] - 1] + k]; ++k);
      he[ra[j]] = k, k = max(0, k - 1);
    }
  }
  int sa[MAXN], ra[MAXN], he[MAXN];
  void build(const string &s) {
    int n = SZ(s);
    fill_n(sa, n, 0), fill_n(ra, n, 0),
      fill_n(he, n, 0);
    fill_n(box, n, 0), fill_n(tp, n, 0), m = 256;
    make_sa(s, n), make_he(s, n);
  }
};
```

6 Math

6.1 chineseRemainder [0e2467]

```
ll solve(ll x1, ll m1, ll x2, ll m2) {
  ll g = gcd(m1, m2);
  if ((x2 - x1) % g) return -1; // no sol
  m1 /= g;
  m2 /= g;
  pll p = exgcd(m1, m2);
  ll lcm = m1 * m2 * g;
  ll res = p.first * (x2 - x1) * m1 + x1;
  // be careful with overflow
  return (res % lcm + lcm) % lcm;
}
```

6.2 PiCount [29fb4b]

```
ll PrimeCount(ll n) { // n ~ 10^13 => < 2s
  if (n <= 1) return 0;
```

```
  int v = sqrt(n), s = (v + 1) / 2, pc = 0;
  vector<int> smalls(v + 1), skip(v + 1), roughs(s);
  vector<ll> larges(s);
  for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
  for (int i = 0; i < s; ++i) {
    roughs[i] = 2 * i + 1;
    larges[i] = (n / (2 * i + 1) + 1) / 2;
  }
  for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
      int q = p * p;
      ++pc;
      if (1LL * q * q > n) break;
      skip[p] = 1;
      for (int i = q; i <= v; i += 2 * p) skip[i] = 1;
      int ns = 0;
      for (int k = 0; k < s; ++k) {
        int i = roughs[k];
        if (skip[i]) continue;
        ll d = 1LL * i * p;
        larges[ns] = larges[k] -
          (d <= v ? larges[smalls[d] - pc]
                  : smalls[n / d]) +
          pc;
        roughs[ns++] = i;
      }
      s = ns;
      for (int j = v / p; j >= p; --j) {
        int c = smalls[j] - pc,
            e = min(j * p + p, v + 1);
        for (int i = j * p; i < e; ++i) smalls[i] -= c;
      }
    }
  }
  for (int k = 1; k < s; ++k) {
    const ll m = n / roughs[k];
    ll t = larges[k] - (pc + k - 1);
    for (int l = 1; l < k; ++l) {
      int p = roughs[l];
      if (1LL * p * p > m) break;
      t -= smalls[m / p] - (pc + l - 1);
    }
    larges[0] -= t;
  }
  return larges[0];
}
```

6.3 numbers

- Bernoulli numbers
  $B_0-1, B_1^{\pm}=\pm\frac{1}{2}, B_2=\frac{1}{6}, B_3=0$
  $\sum_{j=0}^{m}\binom{m+1}{j}B_j=0$, EGF is $B(x)=\frac{x}{e^x-1}=\sum_{n=0}^{\infty}B_n\frac{x^n}{n!}$.
  $S_m(n)=\sum_{k=1}^{n}k^m=\frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k^+n^{m+1-k}$
- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.
  $S(n,k)=S(n-1,k-1)+kS(n-1,k), S(n,1)=S(n,n)=1$
  $S(n,k)=\frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n$
  $x^n=\sum_{i=0}^{n}S(n,i)(x)_i$
- Pentagonal number theorem
  $\prod_{n=1}^{\infty}(1-x^n)=1+\sum_{k=1}^{\infty}(-1)^k\left(x^{k(3k+1)/2}+x^{k(3k-1)/2}\right)$
- Catalan numbers
  $C_n^{(k)}=\frac{1}{(k-1)n+1}\binom{kn}{n}$
  $C^{(k)}(x)=1+x[C^{(k)}(x)]^k$
- Eulerian numbers
  Number of permutations $\pi\in S_n$ in which exactly k elements are greater than the previous element. $k\,j$:s s.t. $\pi(j)>\pi(j+1)$, $k+1\,j$:s s.t. $\pi(j)\geq j$, $k\,j$:s s.t. $\pi(j)>j$.
  $E(n,k)=(n-k)E(n-1,k-1)+(k+1)E(n-1,k)$
  $E(n,0)=E(n,n-1)=1$
  $E(n,k)=\sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$

6.4 Estimation

n	2	3	4	5	6	7	8	9	20	30	40	50	100
$p(n)$	2	3	5	7	11	15	22	30	627	5604	4e4	2e5	2e8

n	100	1e3	1e6	1e9	1e12	1e15	1e18
$d(i)$	12	32	240	1344	6720	26880	103680

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\binom{2n}{n}$	2	6	20	70	252	924	3432	12870	48620	184756	7e5	2e6	1e7	4e7	1.5e8

n	2	3	4	5	6	7	8	9	10	11	12	13
B_n	2	5	15	52	203	877	4140	21147	115975	7e5	4e6	3e7

6.5 floor sum [f931f3]

```
ll floor_sum(ll n, ll m, ll a, ll b) {
  ll ans = 0;
  if (a >= m) ans += (n - 1) * n * (a / m) / 2, a %= m;
  if (b >= m) ans += n * (b / m), b %= m;
  ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
  if (y_max == 0) return ans;
  ans += (n - (x_max + a - 1) / a) * y_max;
  ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
  return ans;
} // sum^{n-1}_0 floor((a * i + b) / m) in log(n + m +
  // a + b)
```

6.6 QuadraticResidue [0b50c4]

```
int Jacobi(int a, int m) {
  int s = 1;
  for (; m > 1;) {
    a %= m;
    if (a == 0) return 0;
    const int r = __builtin_ctz(a);
    if ((r & 1) && ((m + 2) & 4)) s = -s;
    a >>= r;
    if (a & m & 2) s = -s;
    swap(a, m);
  }
  return s;
}

int QuadraticResidue(int a, int p) {
  if (p == 2) return a & 1;
  const int jc = Jacobi(a, p);
  if (jc == 0) return 0;
  if (jc == -1) return -1;
  int b, d;
  for (;;) {
    b = rand() % p;
    d = (1LL * b * b + p - a) % p;
    if (Jacobi(d, p) == -1) break;
  }
  int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
  for (int e = (1LL + p) >> 1; e; e >>= 1) {
    if (e & 1) {
      tmp = (1LL * g0 * f0 +
             1LL * d * (1LL * g1 * f1 % p)) %
        p;
      g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
      g0 = tmp;
    }
    tmp =
      (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) %
      p;
    f1 = (2LL * f0 * f1) % p;
    f0 = tmp;
  }
  return g0;
}
```

6.7 floor enumeration [fc55c8]

```
// enumerating x = floor(n / i), [l, r]
for (int l = 1, r; l <= n; l = r + 1) {
  int x = n / l;
  r = n / x;
}
```

6.8 ax+by=gcd [43bd81]

```
pll exgcd(ll a, ll b) {
  if (b == 0) return pll(1, 0);
  ll p = a / b;
  pll q = exgcd(b, a % b);
  return pll(q.Y, q.X - q.Y * p);
}
/* ax+by=res, let x be minimum non-negative
g, p = gcd(a, b), exgcd(a, b) * res / g
if p.X < 0: t = (abs(p.X) + b / g - 1) / (b / g)
else: t = -(p.X / (b / g))
p += (b / g, -a / g) * t */
```

6.9 cantor expansion [2d801a]

```
#define MAXN 11
int factorial[MAXN];
inline void init() {
  factorial[0] = 1;
  for (int i = 1; i <= MAXN; ++i) {
```

```
    factorial[i] = factorial[i - 1] * i;
  }
}
inline int encode(const std::vector<int> &s) {
  int n = s.size(), res = 0;
  for (int i = 0; i < n; ++i) {
    int t = 0;
    for (int j = i + 1; j < n; ++j) {
      if (s[j] < s[i]) ++t;
    }
    res += t * factorial[n - i - 1];
  }
  return res;
}
inline std::vector<int> decode(int a, int n) {
  std::vector<int> res;
  std::vector<bool> vis(n, 0);
  for (int i = n - 1; i >= 0; --i) {
    int t = a / factorial[i], j;
    for (j = 0; j < n; ++j) {
      if (!vis[j]) {
        if (t == 0) break;
        --t;
      }
    }
    res.push_back(j);
    vis[j] = 1;
    a %= factorial[i];
  }
  return res;
}
```

6.10 Generating function

- Ordinary Generating Function $A(x)=\sum_{i\geq 0}a_ix^i$
  - $A(rx)\Rightarrow r^na_n$
  - $A(x)+B(x)\Rightarrow a_n+b_n$
  - $A(x)B(x)\Rightarrow \sum_{i=0}^{n}a_ib_{n-i}$
  - $A(x)^k\Rightarrow \sum_{i_1+i_2+\cdots+i_k=n}a_{i_1}a_{i_2}...a_{i_k}$
  - $xA(x)'\Rightarrow na_n$
  - $\frac{A(x)}{1-x}\Rightarrow \sum_{i=0}^{n}a_i$
- Exponential Generating Function $A(x)=\sum_{i\geq 0}\frac{a_i}{i!}x_i$
  - $A(x)+B(x)\Rightarrow a_n+b_n$
  - $A^{(k)}(x)\Rightarrow a_{n+k}$
  - $A(x)B(x)\Rightarrow \sum_{i=0}^{n}\binom{n}{i}a_ib_{n-i}$
  - $A(x)^k\Rightarrow \sum_{i_1+i_2+\cdots+i_k=n}\binom{n}{i_1,i_2,\ldots,i_k}a_{i_1}a_{i_2}...a_{i_k}$
  - $xA(x)\Rightarrow na_n$
- Special Generating Function
  - $(1+x)^n=\sum_{i\geq 0}\binom{n}{i}x^i$
  - $\frac{1}{(1-x)^n}=\sum_{i\geq 0}\binom{i}{n-1}x^i$

6.11 Fraction [666134]

```
struct fraction {
  ll n, d;
  fraction(const ll &_n = 0, const ll &_d = 1)
    : n(_n), d(_d) {
    ll t = gcd(n, d);
    n /= t, d /= t;
    if (d < 0) n = -n, d = -d;
  }
  fraction operator-() const {
    return fraction(-n, d);
  }
  fraction operator+(const fraction &b) const {
    return fraction(n * b.d + b.n * d, d * b.d);
  }
  fraction operator-(const fraction &b) const {
    return fraction(n * b.d - b.n * d, d * b.d);
  }
  fraction operator*(const fraction &b) const {
    return fraction(n * b.n, d * b.d);
  }
  fraction operator/(const fraction &b) const {
    return fraction(n * b.d, d * b.n);
  }
  void print() {
    cout << n;
    if (d != 1) cout << "/" << d;
  }
};
```

6.12 Gaussian gcd [616465]

```
cpx gaussian_gcd(cpx a, cpx b) {
#define rnd(a, b)                                   \
  ((a >= 0 ? a * 2 + b : a * 2 - b) / (b * 2))
  ll c = a.real() * b.real() + a.imag() * b.imag();
  ll d = a.imag() * b.real() - a.real() * b.imag();
  ll r = b.real() * b.real() + b.imag() * b.imag();
  if (c % r == 0 && d % r == 0) return b;
  return gaussian_gcd(
    b, a - cpx(rnd(c, r), rnd(d, r)) * b);
}
```

6.13 Theorem

- Cramer's rule
$$\begin{matrix} ax+by=e \\ cx+dy=f \end{matrix}\Rightarrow \begin{matrix} x=\frac{ed-bf}{ad-bc} \\ y=\frac{af-ec}{ad-bc} \end{matrix}$$
- Vandermonde's Identity
$$C(n+m,k)=\sum_{i=0}^{k}C(n,i)C(m,k-i)$$
- Kirchhoff's Theorem
  Denote L be a $n\times n$ matrix as the Laplacian matrix of graph G, where $L_{ii}=d(i)$, $L_{ij}=-c$ where c is the number of edge (i,j) in G.
  - The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
  - The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.
- Tutte's Matrix
  Let D be a $n\times n$ matrix, where $d_{ij}=x_{ij}$ (x_{ij} is chosen uniformly at random) if $i<j$ and $(i,j)\in E$, otherwise $d_{ij}=-d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.
- Cayley's Formula
  - Given a degree sequence $d_1,d_2,\ldots,d_n$ for each *labeled* vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
  - Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1,2,\ldots,k$ belong to different components. Then $T_{n,k}=kn^{n-k-1}$.
- Erdős–Gallai theorem
  A sequence of nonnegative integers $d_1\geq\cdots\geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1+\cdots+d_n$ is even and $\sum_{i-1}^{k}d_i\leq k(k-1)+\sum_{i=k+1}^{n}\min(d_i,k)$ holds for every $1\leq k\leq n$.
- Gale–Ryser theorem
  A pair of sequences of nonnegative integers $a_1\geq\cdots\geq a_n$ and $b_1,\ldots,b_n$ is bigraphic if and only if $\sum_{i=1}^{n}a_i=\sum_{i=1}^{n}b_i$ and $\sum_{i=1}^{k}a_i\leq\sum_{i=1}^{n}\min(b_i,k)$ holds for every $1\leq k\leq n$.
- Fulkerson–Chen–Anstee theorem
  A sequence $(a_1,b_1),\ldots,(a_n,b_n)$ of nonnegative integer pairs with $a_1\geq\cdots\geq a_n$ is digraphic if and only if $\sum_{i=1}^{n}a_i=\sum_{i=1}^{n}b_i$ and $\sum_{i=1}^{k}a_i\leq\sum_{i=1}^{k}\min(b_i,k-1)+\sum_{i=k+1}^{n}\min(b_i,k)$ holds for every $1\leq k\leq n$.
- Pick's theorem
  For simple polygon, when points are all integer, we have $A=\#\{\text{lattice points in the interior}\}+\frac{\#\{\text{lattice points on the boundary}\}}{2}-1$.
- Möbius inversion formula
  - $f(n)=\sum_{d|n}g(d)\Leftrightarrow g(n)=\sum_{d|n}\mu(d)f(\frac{n}{d})$
  - $f(n)=\sum_{n|d}g(d)\Leftrightarrow g(n)=\sum_{n|d}\mu(\frac{d}{n})f(d)$
- Spherical cap
  - A portion of a sphere cut off by a plane.
  - r: sphere radius, a: radius of the base of the cap, h: height of the cap, θ: $\arcsin(a/r)$.
  - Volume $=\pi h^2(3r-h)/3=\pi h(3a^2+h^2)/6=\pi r^3(2+\cos\theta)(1-\cos\theta)^2/3$.
  - Area $=2\pi rh=\pi(a^2+h^2)=2\pi r^2(1-\cos\theta)$.
- Lagrange multiplier
  - Optimize $f(x_1,\ldots,x_n)$ when k constraints $g_i(x_1,\ldots,x_n)=0$.
  - Lagrangian function $\mathcal{L}(x_1,\ldots,x_n,\lambda_1,\ldots,\lambda_k)=f(x_1,\ldots,x_n)-\sum_{i=1}^{k}\lambda_ig_i(x_1,\ldots,x_n)$.
  - The solution corresponding to the original constrained optimization is always a saddle point of the Lagrangian function.
- Nearest points of two skew lines
  - Line 1: $\boldsymbol{v}_1=\boldsymbol{p}_1+t_1\boldsymbol{d}_1$
  - Line 2: $\boldsymbol{v}_2=\boldsymbol{p}_2+t_2\boldsymbol{d}_2$
  - $\boldsymbol{n}=\boldsymbol{d}_1\times\boldsymbol{d}_2$

- $\boldsymbol{n}_1 = \boldsymbol{d}_1 \times \boldsymbol{n}$
- $\boldsymbol{n}_2 = \boldsymbol{d}_2 \times \boldsymbol{n}$
- $\boldsymbol{c}_1 = \boldsymbol{p}_1 + \frac{(\boldsymbol{p}_2 - \boldsymbol{p}_1)\cdot \boldsymbol{n}_2}{\boldsymbol{d}_1 \cdot \boldsymbol{n}_2}\boldsymbol{d}_1$
- $\boldsymbol{c}_2 = \boldsymbol{p}_2 + \frac{(\boldsymbol{p}_1 - \boldsymbol{p}_2)\cdot \boldsymbol{n}_1}{\boldsymbol{d}_2 \cdot \boldsymbol{n}_1}\boldsymbol{d}_2$

- Derivatives/Integrals

  Integration by parts: $\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$

$$\left|\begin{array}{l} \frac{d}{dx}\sin^{-1}x = \frac{1}{\sqrt{1-x^2}} \\ \frac{d}{dx}\tan x = 1+\tan^2 x \\ \int e^{-x^2} = \frac{\sqrt{\pi}}{2}\text{erf}(x) \end{array}\right| \left|\begin{array}{l} \frac{d}{dx}\cos^{-1}x = -\frac{1}{\sqrt{1-x^2}} \\ \int \tan ax = -\frac{\ln|\cos ax|}{a} \\ \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1) \end{array}\right| \left|\begin{array}{l} \frac{d}{dx}\tan^{-1}x = \frac{1}{1+x^2} \end{array}\right|$$

$$\int \sqrt{a^2+x^2} = \frac{1}{2}\left(x\sqrt{a^2+x^2} + a^2 \text{asinh}(x/a)\right)$$

- Spherical Coordinate

$$(x,y,z) = (r\sin\theta\cos\phi, r\sin\theta\sin\phi, r\cos\theta)$$

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

- Rotation Matrix

$$M(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

6.14 Determinant [a4d696]

```cpp
struct Matrix {
  int n, m;
  ll M[MAXN][MAXN];
  int row_swap(int i, int j) {
    if (i == j) return 0;
    for (int k = 0; k < m; ++k) swap(M[i][k], M[j][k]);
    return 1;
  }
  ll det() { // return the number of swaps
    int rt = 0;
    for (int i = 0; i < n; ++i) {
      int piv = i;
      while (piv < n && !M[piv][i]) ++piv;
      if (piv == n) continue;
      rt += row_swap(i, piv);
      for (int j = i + 1; j < n; ++j) {
        while (M[j][i]) {
          int tmp = P - M[i][i] / M[j][i];
          for (int k = i; k < m; ++k)
            M[i][k] = (M[j][k] * tmp + M[i][k]) % P;
          rt += row_swap(i, j);
        }
      }
    }
    rt = (rt & 1) ? P - 1 : 1;
    for (int i = 0; i < n; ++i) rt = rt * M[i][i] % P;
    return rt;
    // round(rt) if using double to cal. int. det
  }
};
```

6.15 ModMin [05065e]

```cpp
// min{k | l <= ((ak) mod m) <= r}, no solution -> -1
ll mod_min(ll a, ll m, ll l, ll r) {
  if (a == 0) return l ? -1 : 0;
  if (ll k = (l + a - 1) / a; k * a <= r) return k;
  ll b = m / a, c = m % a;
  if (ll y = mod_min(c, a, a - r % a, a - l % a))
    return (l + y * c + a - 1) / a + y * b;
  return -1;
}
```

6.16 Primes [2464ae]

```cpp
/* 12721 13331 14341 75577 123457 222557 556679 999983
 * 1097774749 1076767633 100102021 999997771 1001010013
 * 1000512343 987654361 999991231 999888733 98789101
 * 987777733 999991921 1010101333 1010102101
 * 1000000000039 1000000000000037 2305843009213693951
 * 4611686018427387847 9223372036854775783
 * 18446744073709551557 */
```

6.17 Pollard Rho [a5802e]

```cpp
map<ll, int> cnt;
void PollardRho(ll n) {
  if (n == 1) return;
  if (prime(n)) return ++cnt[n], void();
  if (n % 2 == 0)
    return PollardRho(n / 2), ++cnt[2], void();
  ll x = 2, y = 2, d = 1, p = 1;
#define f(x, n, p) ((mul(x, x, n) + p) % n)
  while (true) {
    if (d != n && d != 1) {
      PollardRho(n / d);
      PollardRho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p), y = f(f(y, n, p), n, p);
    d = gcd(abs(x - y), n);
  }
}
```

6.18 Simultaneous Equations [b8b03f]

```cpp
struct matrix { // m variables, n equations
  int n, m;
  fraction M[MAXN][MAXN + 1], sol[MAXN];
  int solve() { //-1: inconsistent, >= 0: rank
    for (int i = 0; i < n; ++i) {
      int piv = 0;
      while (piv < m && !M[i][piv].n) ++piv;
      if (piv == m) continue;
      for (int j = 0; j < n; ++j) {
        if (i == j) continue;
        fraction tmp = -M[j][piv] / M[i][piv];
        for (int k = 0; k <= m; ++k)
          M[j][k] = tmp * M[i][k] + M[j][k];
      }
    }
    int rank = 0;
    for (int i = 0; i < n; ++i) {
      int piv = 0;
      while (piv < m && !M[i][piv].n) ++piv;
      if (piv == m && M[i][m].n) return -1;
      else if (piv < m)
        ++rank, sol[piv] = M[i][m] / M[i][piv];
    }
    return rank;
  }
};
```

6.19 Big number [1c17ab]

```cpp
template <typename T>
inline string to_string(const T &x) {
  stringstream ss;
  return ss << x, ss.str();
}
struct bigN : vector<ll> {
  const static int base = 1000000000,
                   width = log10(base);
  bool negative;
  bigN(const_iterator a, const_iterator b)
    : vector<ll>(a, b) {}
  bigN(string s) {
    if (s.empty()) return;
    if (s[0] == '-') negative = 1, s = s.substr(1);
    else negative = 0;
    for (int i = int(s.size()) - 1; i >= 0;
         i -= width) {
      ll t = 0;
      for (int j = max(0, i - width + 1); j <= i; ++j)
        t = t * 10 + s[j] - '0';
      push_back(t);
    }
    trim();
  }
  template <typename T>
  bigN(const T &x) : bigN(to_string(x)) {}
  bigN() : negative(0) {}
  void trim() {
    while (size() && !back()) pop_back();
    if (empty()) negative = 0;
  }
  void carry(int _base = base) {
    for (size_t i = 0; i < size(); ++i) {
      if (at(i) >= 0 && at(i) < _base) continue;
```

```
      if (i + 1u == size()) push_back(0);
      int r = at(i) % _base;
      if (r < 0) r += _base;
      at(i + 1) += (at(i) - r) / _base, at(i) = r;
    }
  }
  int abscmp(const bigN &b) const {
    if (size() > b.size()) return 1;
    if (size() < b.size()) return -1;
    for (int i = int(size()) - 1; i >= 0; --i) {
      if (at(i) > b[i]) return 1;
      if (at(i) < b[i]) return -1;
    }
    return 0;
  }
  int cmp(const bigN &b) const {
    if (negative != b.negative)
      return negative ? -1 : 1;
    return negative ? -abscmp(b) : abscmp(b);
  }
  bool operator<(const bigN &b) const {
    return cmp(b) < 0;
  }
  bool operator>(const bigN &b) const {
    return cmp(b) > 0;
  }
  bool operator<=(const bigN &b) const {
    return cmp(b) <= 0;
  }
  bool operator>=(const bigN &b) const {
    return cmp(b) >= 0;
  }
  bool operator==(const bigN &b) const {
    return !cmp(b);
  }
  bool operator!=(const bigN &b) const {
    return cmp(b) != 0;
  }
  bigN abs() const {
    bigN res = *this;
    return res.negative = 0, res;
  }
  bigN operator-() const {
    bigN res = *this;
    return res.negative = !negative, res.trim(), res;
  }
  bigN operator+(const bigN &b) const {
    if (negative) return -(-(*this) + (-b));
    if (b.negative) return *this - (-b);
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (size_t i = 0; i < b.size(); ++i)
      res[i] += b[i];
    return res.carry(), res.trim(), res;
  }
  bigN operator-(const bigN &b) const {
    if (negative) return -(-(*this) - (-b));
    if (b.negative) return *this + (-b);
    if (abscmp(b) < 0) return -(b - (*this));
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (size_t i = 0; i < b.size(); ++i)
      res[i] -= b[i];
    return res.carry(), res.trim(), res;
  }
  bigN operator*(const bigN &b) const {
    bigN res;
    res.negative = negative != b.negative;
    res.resize(size() + b.size());
    for (size_t i = 0; i < size(); ++i)
      for (size_t j = 0; j < b.size(); ++j)
        if ((res[i + j] += at(i) * b[j]) >= base) {
          res[i + j + 1] += res[i + j] / base;
          res[i + j] %= base;
        } // ¾ª k¥◆carry · | · ¸¦◆
    return res.trim(), res;
  }
  bigN operator/(const bigN &b) const {
    int norm = base / (b.back() + 1);
    bigN x = abs() * norm;
    bigN y = b.abs() * norm;
    bigN q, r;
    q.resize(x.size());
    for (int i = int(x.size()) - 1; i >= 0; --i) {
      r = r * base + x[i];
      int s1 = r.size() <= y.size() ? 0 : r[y.size()];
      int s2 =
        r.size() < y.size() ? 0 : r[y.size() - 1];
      int d = (ll(base) * s1 + s2) / y.back();
      r = r - y * d;
      while (r.negative) r = r + y, --d;
      q[i] = d;
    }
    q.negative = negative != b.negative;
    return q.trim(), q;
  }
  bigN operator%(const bigN &b) const {
    return *this - (*this / b) * b;
  }
  friend istream &operator>>(istream &ss, bigN &b) {
    string s;
    return ss >> s, b = s, ss;
  }
  friend ostream &operator<<(
    ostream &ss, const bigN &b) {
    if (b.negative) ss << '-';
    ss << (b.empty() ? 0 : b.back());
    for (int i = int(b.size()) - 2; i >= 0; --i)
      ss << setw(width) << setfill('0') << b[i];
    return ss;
  }
  template <typename T> operator T() {
    stringstream ss;
    ss << *this;
    T res;
    return ss >> res, res;
  }
};
```

6.20 Euclidean

- $m = \lfloor \frac{an+b}{c} \rfloor$
- Time complexity: $O(\log n)$

$$f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)}{2} + \lfloor \frac{b}{c} \rfloor \cdot (n+1) & \\ \quad + f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm - f(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$

$$g(a,b,c,n) = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor \cdot \frac{n(n+1)}{2} & \\ \quad + g(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1) & \\ \quad - h(c, c-b-1, a, m-1)), & \text{otherwise} \end{cases}$$

$$h(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor^2 \cdot (n+1) & \\ \quad + \lfloor \frac{a}{c} \rfloor \cdot \lfloor \frac{b}{c} \rfloor \cdot n(n+1) & \\ \quad + h(a \bmod c, b \bmod c, c, n) & \\ \quad + 2 \lfloor \frac{a}{c} \rfloor \cdot g(a \bmod c, b \bmod c, c, n) & \\ \quad + 2 \lfloor \frac{b}{c} \rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) & \\ \quad - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

6.21 Miller Rabin [969881]

```
// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : primes <= 13
// n < 2^64               7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool Miller_Rabin(ll a, ll n) {
  if ((a = a % n) == 0) return 1;
  if (n % 2 == 0) return n == 2;
  ll tmp = (n - 1) / ((n - 1) & (1 - n));
  ll t = __lg(((n - 1) & (1 - n))), x = 1;
  for (; tmp; tmp >>= 1, a = mul(a, a, n))
    if (tmp & 1) x = mul(x, a, n);
  if (x == 1 || x == n - 1) return 1;
  while (--t)
    if ((x = mul(x, x, n)) == n - 1) return 1;
  return 0;
}
```

6.22 Berlekamp-Massey [cdb091]

```cpp
template <typename T>
vector<T> BerlekampMassey(const vector<T> &output) {
  vector<T> d(SZ(output) + 1), me, he;
  for (int f = 0, i = 1; i <= SZ(output); ++i) {
    for (int j = 0; j < SZ(me); ++j)
      d[i] += output[i - j - 2] * me[j];
    if ((d[i] -= output[i - 1]) == 0) continue;
    if (me.empty()) {
      me.resize(f = i);
      continue;
    }
    vector<T> o(i - f - 1);
    T k = -d[i] / d[f];
    o.pb(-k);
    for (T x : he) o.pb(x * k);
    o.resize(max(SZ(o), SZ(me)));
    for (int j = 0; j < SZ(me); ++j) o[j] += me[j];
    if (i - f + SZ(he) >= SZ(me)) he = me, f = i;
    me = o;
  }
  return me;
}
```

6.23 floor ceil [f84849]

```cpp
int floor(int a, int b) {
  return a / b - (a % b && (a < 0) ^ (b < 0));
}
int ceil(int a, int b) {
  return a / b + (a % b && (a < 0) ^ (b > 0));
}
```

6.24 fac no p [86ad89]

```cpp
// O(p^k + log^2 n), pk = p^k
ll prod[MAXP];
ll fac_no_p(ll n, ll p, ll pk) {
  prod[0] = 1;
  for (int i = 1; i <= pk; ++i)
    if (i % p) prod[i] = prod[i - 1] * i % pk;
    else prod[i] = prod[i - 1];
  ll rt = 1;
  for (; n; n /= p) {
    rt = rt * mpow(prod[pk], n / pk, pk) % pk;
    rt = rt * prod[n % pk] % pk;
  }
  return rt;
} // (n! without factor p) % p^k
```

6.25 DiscreteLog [21f791]

```cpp
int DiscreteLog(int s, int x, int y, int m) {
  constexpr int kStep = 32000;
  unordered_map<int, int> p;
  int b = 1;
  for (int i = 0; i < kStep; ++i) {
    p[y] = i;
    y = 1LL * y * x % m;
    b = 1LL * b * x % m;
  }
  for (int i = 0; i < m + 10; i += kStep) {
    s = 1LL * s * b % m;
    if (p.find(s) != p.end()) return i + kStep - p[s];
  }
  return -1;
}
int DiscreteLog(int x, int y, int m) {
  if (m == 1) return 0;
  int s = 1;
  for (int i = 0; i < 100; ++i) {
    if (s == y) return i;
    s = 1LL * s * x % m;
  }
  if (s == y) return 100;
  int p = 100 + DiscreteLog(s, x, y, m);
  if (fpow(x, p, m) != y) return -1;
  return p;
}
```

6.26 SimplexConstruction

Primal	Dual
Maximize c^Tx s.t. $Ax \le b, x \ge 0$	Minimize b^Ty s.t. $A^Ty \ge c, y \ge 0$
Maximize c^Tx s.t. $Ax \le b$	Minimize b^Ty s.t. $A^Ty = c, y \ge 0$
Maximize c^Tx s.t. $Ax = b, x \ge 0$	Minimize b^Ty s.t. $A^Ty \ge c$

$\bar{x}$ and $\bar{y}$ are optimal if and only if for all $i \in [1,n]$, either $\bar{x}_i = 0$ or $\sum_{j=1}^{m} A_{ji}\bar{y}_j = c_i$ holds and for all $i \in [1,m]$ either $\bar{y}_i = 0$ or $\sum_{j=1}^{n} A_{ij}\bar{x}_j = b_j$ holds.

1. In case of minimization, let $c_i' = -c_i$
2. $\sum_{1\le i\le n} A_{ji}x_i \ge b_j \to \sum_{1\le i\le n} -A_{ji}x_i \le -b_j$
3. $\sum_{1\le i\le n} A_{ji}x_i = b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \le b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \ge b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x_i'$

6.27 Simplex Algorithm [ad99b3]

```cpp
const int MAXN = 11000, MAXM = 405;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM];
double d[MAXN][MAXM], x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(int n, int m) {
  ++m;
  fill_n(d[n], m + 1, 0);
  fill_n(d[n + 1], m + 1, 0);
  iota(ix, ix + n + m, 0);
  int r = n, s = m - 1;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1;
    d[i][m] = b[i];
    if (d[r][m] > d[i][m]) r = i;
  }
  copy_n(c, m - 1, d[n]);
  d[n + 1][m - 1] = -1;
  for (double dd;;) {
    if (r < n) {
      swap(ix[s], ix[r + m]);
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j)
        if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i)
        if (i != r) {
          for (int j = 0; j <= m; ++j)
            if (j != s) d[i][j] += d[r][j] * d[i][s];
          d[i][s] *= d[r][s];
        }
    }
    r = s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 || ix[s] > ix[j]) {
        if (d[n + 1][j] > eps ||
          (d[n + 1][j] > -eps && d[n][j] > eps))
          s = j;
      }
    if (s < 0) break;
    for (int i = 0; i < n; ++i)
      if (d[i][s] < -eps) {
        if (r < 0 ||
          (dd = d[r][m] / d[r][s] -
              d[i][m] / d[i][s]) < -eps ||
          (dd < eps && ix[r + m] > ix[i + m]))
          r = i;
      }
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  fill_n(x, m, 0);
  for (int i = m; i < n + m;
       ++i) { // the missing enumerated x[i] = 0
    if (ix[i] < m - 1) {
      ans += d[i - m][m] * c[ix[i]];
      x[ix[i]] = d[i - m][m];
    }
  }
  return ans;
}
```

6.28 SchreierSims [c5604c]

```cpp
namespace schreier {
int n;
vector<vector<vector<int>>> bkts, binv;
vector<vector<int>> lk;
vector<int> operator*(
  const vector<int> &a, const vector<int> &b) {
  vector<int> res(SZ(a));
  for (int i = 0; i < SZ(a); ++i) res[i] = b[a[i]];
```

```
  return res;
}
vector<int> inv(const vector<int> &a) {
  vector<int> res(SZ(a));
  for (int i = 0; i < SZ(a); ++i) res[a[i]] = i;
  return res;
}
int filter(const vector<int> &g, bool add = true) {
  n = SZ(bkts);
  vector<int> p = g;
  for (int i = 0; i < n; ++i) {
    assert(p[i] >= 0 && p[i] < SZ(lk[i]));
    if (lk[i][p[i]] == -1) {
      if (add) {
        bkts[i].pb(p);
        binv[i].pb(inv(p));
        lk[i][p[i]] = SZ(bkts[i]) - 1;
      }
      return i;
    }
    p = p * binv[i][lk[i][p[i]]];
  }
  return -1;
}
bool inside(const vector<int> &g) {
  return filter(g, false) == -1;
}
void solve(const vector<vector<int>> &gen, int _n) {
  n = _n;
  bkts.clear(), bkts.resize(n);
  binv.clear(), binv.resize(n);
  lk.clear(), lk.resize(n);
  vector<int> iden(n);
  iota(iden.begin(), iden.end(), 0);
  for (int i = 0; i < n; ++i) {
    lk[i].resize(n, -1);
    bkts[i].pb(iden);
    binv[i].pb(iden);
    lk[i][i] = 0;
  }
  for (int i = 0; i < SZ(gen); ++i) filter(gen[i]);
  queue<pair<pii, pii>> upd;
  for (int i = 0; i < n; ++i)
    for (int j = i; j < n; ++j)
      for (int k = 0; k < SZ(bkts[i]); ++k)
        for (int l = 0; l < SZ(bkts[j]); ++l)
          upd.emplace(pii(i, k), pii(j, l));
  while (!upd.empty()) {
    auto a = upd.front().X;
    auto b = upd.front().Y;
    upd.pop();
    int res = filter(bkts[a.X][a.Y] * bkts[b.X][b.Y]);
    if (res == -1) continue;
    pii pr = pii(res, SZ(bkts[res]) - 1);
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < SZ(bkts[i]); ++j) {
        if (i <= res) upd.emplace(pii(i, j), pr);
        if (res <= i) upd.emplace(pr, pii(i, j));
      }
  }
}
ll size() {
  ll res = 1;
  for (int i = 0; i < n; ++i) res = res * SZ(bkts[i]);
  return res;
}
} // namespace schreier
```

7 Polynomial

7.1 Polynomial Operation [ddbb6e]

```
#define fi(s, n)                                                \
  for (int i = (int)(s); i < (int)(n); ++i)
template <int MAXN, ll P, ll RT> // MAXN = 2^k
struct Poly : vector<ll> { // coefficients in [0, P)
  using vector<ll>::vector;
  static NTT<MAXN, P, RT> ntt;
  int n() const { return (int)size(); } // n() >= 1
  Poly(const Poly &p, int m) : vector<ll>(m) {
    copy_n(p.data(), min(p.n(), m), data());
  }
  Poly &irev() {
    return reverse(data(), data() + n()), *this;
  }
  Poly &isz(int m) { return resize(m), *this; }
  Poly &iadd(const Poly &rhs) { // n() == rhs.n()
    fi(0, n()) if (((*this)[i] += rhs[i]) >= P)(
      *this)[i] -= P;
    return *this;
  }
  Poly &imul(ll k) {
    fi(0, n())(*this)[i] = (*this)[i] * k % P;
    return *this;
  }
  Poly Mul(const Poly &rhs) const {
    int m = 1;
    while (m < n() + rhs.n() - 1) m <<= 1;
    Poly X(*this, m), Y(rhs, m);
    ntt(X.data(), m), ntt(Y.data(), m);
    fi(0, m) X[i] = X[i] * Y[i] % P;
    ntt(X.data(), m, true);
    return X.isz(n() + rhs.n() - 1);
  }
  Poly Inv() const { // (*this)[0] != 0, 1e5/95ms
    if (n() == 1) return {ntt.minv((*this)[0])};
    int m = 1;
    while (m < n() * 2) m <<= 1;
    Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(m);
    Poly Y(*this, m);
    ntt(Xi.data(), m), ntt(Y.data(), m);
    fi(0, m) {
      Xi[i] *= (2 - Xi[i] * Y[i]) % P;
      if ((Xi[i] %= P) < 0) Xi[i] += P;
    }
    ntt(Xi.data(), m, true);
    return Xi.isz(n());
  }
  Poly Sqrt()
    const { // Jacobi((*this)[0], P) = 1, 1e5/235ms
    if (n() == 1)
      return {QuadraticResidue((*this)[0], P)};
    Poly X =
      Poly(*this, (n() + 1) / 2).Sqrt().isz(n());
    return X.iadd(Mul(X.Inv()).isz(n()))
      .imul(P / 2 + 1);
  }
  pair<Poly, Poly> DivMod(
    const Poly &rhs) const { // (rhs.)back() != 0
    if (n() < rhs.n()) return {{0}, *this};
    const int m = n() - rhs.n() + 1;
    Poly X(rhs);
    X.irev().isz(m);
    Poly Y(*this);
    Y.irev().isz(m);
    Poly Q = Y.Mul(X.Inv()).isz(m).irev();
    X = rhs.Mul(Q), Y = *this;
    fi(0, n()) if ((Y[i] -= X[i]) < 0) Y[i] += P;
    return {Q, Y.isz(max(1, rhs.n() - 1))};
  }
  Poly Dx() const {
    Poly ret(n() - 1);
    fi(0, ret.n()) ret[i] =
      (i + 1) * (*this)[i + 1] % P;
    return ret.isz(max(1, ret.n()));
  }
  Poly Sx() const {
    Poly ret(n() + 1);
    fi(0, n()) ret[i + 1] =
      ntt.minv(i + 1) * (*this)[i] % P;
    return ret;
  }
  Poly _tmul(int nn, const Poly &rhs) const {
    Poly Y = Mul(rhs).isz(n() + nn - 1);
    return Poly(Y.data() + n() - 1, Y.data() + Y.n());
  }
  vector<ll> _eval(const vector<ll> &x,
    const vector<Poly> &up) const {
    const int m = (int)x.size();
    if (!m) return {};
    vector<Poly> down(m * 2);
    // down[1] = DivMod(up[1]).second;
    // fi(2, m * 2) down[i] = down[i /
    // 2].DivMod(up[i]).second;
    down[1] =
      Poly(up[1]).irev().isz(n()).Inv().irev()._tmul(
        m, *this);
    fi(2, m * 2) down[i] =
      up[i ^ 1]._tmul(up[i].n() - 1, down[i / 2]);
    vector<ll> y(m);
    fi(0, m) y[i] = down[m + i][0];
    return y;
  }
```

```
  static vector<Poly> _tree1(const vector<ll> &x) {
    const int m = (int)x.size();
    vector<Poly> up(m * 2);
    fi(0, m) up[m + i] = {(x[i] ? P - x[i] : 0), 1};
    for (int i = m - 1; i > 0; --i)
      up[i] = up[i * 2].Mul(up[i * 2 + 1]);
    return up;
  }
  vector<ll> Eval(
    const vector<ll> &x) const { // 1e5, 1s
    auto up = _tree1(x);
    return _eval(x, up);
  }
  static Poly Interpolate(const vector<ll> &x,
    const vector<ll> &y) { // 1e5, 1.4s
    const int m = (int)x.size();
    vector<Poly> up = _tree1(x), down(m * 2);
    vector<ll> z = up[1].Dx()._eval(x, up);
    fi(0, m) z[i] = y[i] * ntt.minv(z[i]) % P;
    fi(0, m) down[m + i] = {z[i]};
    for (int i = m - 1; i > 0; --i)
      down[i] =
        down[i * 2]
          .Mul(up[i * 2 + 1])
          .iadd(down[i * 2 + 1].Mul(up[i * 2]));
    return down[1];
  }
  Poly Ln() const { // (*this)[0] == 1, 1e5/170ms
    return Dx().Mul(Inv()).Sx().isz(n());
  }
  Poly Exp() const { // (*this)[0] == 0, 1e5/360ms
    if (n() == 1) return {1};
    Poly X = Poly(*this, (n() + 1) / 2).Exp().isz(n());
    Poly Y = X.Ln();
    Y[0] = P - 1;
    fi(0, n()) if ((Y[i] = (*this)[i] - Y[i]) < 0)
      Y[i] += P;
    return X.Mul(Y).isz(n());
  }
  // M := P(P - 1). If k >= M, k := k % M + M.
  Poly Pow(ll k) const {
    int nz = 0;
    while (nz < n() && !(*this)[nz]) ++nz;
    if (nz * min(k, (ll)n()) >= n()) return Poly(n());
    if (!k) return Poly(Poly{1}, n());
    Poly X(data() + nz, data() + nz + n() - nz * k);
    const ll c = ntt.mpow(X[0], k % (P - 1));
    return X.Ln()
      .imul(k % P)
      .Exp()
      .imul(c)
      .irev()
      .isz(n())
      .irev();
  }
  static ll LinearRecursion(const vector<ll> &a,
    const vector<ll> &coef,
    ll n) { // a_n = \sum c_j a_(n-j)
    const int k = (int)a.size();
    assert((int)coef.size() == k + 1);
    Poly C(k + 1), W(Poly{1}, k), M = {0, 1};
    fi(1, k + 1) C[k - i] = coef[i] ? P - coef[i] : 0;
    C[k] = 1;
    while (n) {
      if (n % 2) W = W.Mul(M).DivMod(C).second;
      n /= 2, M = M.Mul(M).DivMod(C).second;
    }
    ll ret = 0;
    fi(0, k) ret = (ret + W[i] * a[i]) % P;
    return ret;
  }
};
#undef fi
using Poly_t = Poly<131072 * 2, 998244353, 3>;
template <> decltype(Poly_t::ntt) Poly_t::ntt = {};
```

7.2 Fast Walsh Transform [820c20]

```
/* x: a[j], y: a[j + (L >> 1)]
or: (y += x * op), and: (x += y * op)
xor: (x, y = (x + y) * op, (x - y) * op)
invop: or, and, xor = -1, -1, 1/2 */
void fwt(int *a, int n, int op) { // or
  for (int L = 2; L <= n; L <<= 1)
    for (int i = 0; i < n; i += L)
      for (int j = i; j < i + (L >> 1); ++j)
        a[j + (L >> 1)] += a[j] * op;
}
const int N = 21;
int f[N][1 << N], g[N][1 << N], h[N][1 << N],
  ct[1 << N];
void subset_convolution(
  int *a, int *b, int *c, int L) {
  // c_k = \sum_{i | j = k, i & j = 0} a_i * b_j
  int n = 1 << L;
  for (int i = 1; i < n; ++i)
    ct[i] = ct[i & (i - 1)] + 1;
  for (int i = 0; i < n; ++i)
    f[ct[i]][i] = a[i], g[ct[i]][i] = b[i];
  for (int i = 0; i <= L; ++i)
    fwt(f[i], n, 1), fwt(g[i], n, 1);
  for (int i = 0; i <= L; ++i)
    for (int j = 0; j <= i; ++j)
      for (int x = 0; x < n; ++x)
        h[i][x] += f[j][x] * g[i - j][x];
  for (int i = 0; i <= L; ++i) fwt(h[i], n, -1);
  for (int i = 0; i < n; ++i) c[i] = h[ct[i]][i];
}
```

7.3 Number Theory Transform [9a0ea6]

```
//(2^16)+1, 65537, 3
// 7*17*(2^23)+1, 998244353, 3
// 1255*(2^20)+1, 1315962881, 3
// 51*(2^25)+1, 1711276033, 29
template <int MAXN, ll P, ll RT> // MAXN must be 2^k
struct NTT {
  ll w[MAXN];
  ll mpow(ll a, ll n);
  ll minv(ll a) { return mpow(a, P - 2); }
  NTT() {
    ll dw = mpow(RT, (P - 1) / MAXN);
    w[0] = 1;
    for (int i = 1; i < MAXN; ++i)
      w[i] = w[i - 1] * dw % P;
  }
  void bitrev(ll *a, int n) {
    int i = 0;
    for (int j = 1; j < n - 1; ++j) {
      for (int k = n >> 1; (i ^= k) < k; k >>= 1);
      if (j < i) swap(a[i], a[j]);
    }
  }
  void operator()(
    ll *a, int n, bool inv = false) { // 0 <= a[i] < P
    bitrev(a, n);
    for (int L = 2; L <= n; L <<= 1) {
      int dx = MAXN / L, dl = L >> 1;
      for (int i = 0; i < n; i += L) {
        for (int j = i, x = 0; j < i + dl;
             ++j, x += dx) {
          ll tmp = a[j + dl] * w[x] % P;
          if ((a[j + dl] = a[j] - tmp) < 0)
            a[j + dl] += P;
          if ((a[j] += tmp) >= P) a[j] -= P;
        }
      }
    }
    if (inv) {
      reverse(a + 1, a + n);
      ll invn = minv(n);
      for (int i = 0; i < n; ++i)
        a[i] = a[i] * invn % P;
    }
  }
};
```

7.4 Value Poly [6438ba]

```
struct Poly {
  mint base; // f(x) = poly[x - base]
  vector<mint> poly;
  Poly(mint b = 0, mint x = 0) : base(b), poly(1, x) {}
  mint get_val(const mint &x) {
    if (x >= base && x < base + SZ(poly))
      return poly[x - base];
    mint rt = 0;
    vector<mint> lmul(SZ(poly), 1), rmul(SZ(poly), 1);
    for (int i = 1; i < SZ(poly); ++i)
      lmul[i] = lmul[i - 1] * (x - (base + i - 1));
    for (int i = SZ(poly) - 2; i >= 0; --i)
      rmul[i] = rmul[i + 1] * (x - (base + i + 1));
    for (int i = 0; i < SZ(poly); ++i)
      rt += poly[i] * ifac[i] *
```

```
        inegfac[SZ(poly) - 1 - i] * lmul[i] * rmul[i];
    return rt;
  }
  void raise() { // g(x) = sigma{base:x} f(x)
    if (SZ(poly) == 1 && poly[0] == 0) return;
    mint nw = get_val(base + SZ(poly));
    poly.pb(nw);
    for (int i = 1; i < SZ(poly); ++i)
      poly[i] += poly[i - 1];
  }
};
```

7.5 NTT.2 [6997db]

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int MAXN = 1 << 20;
template <int MOD, int RT> struct Zp {
#define OP(op) static int op(int x, int y)
  OP(add) { return (x += y) >= MOD ? x - MOD : x; }
  OP(sub) { return (x -= y) < 0 ? x + MOD : x; }
  OP(mul) { return int(ll(x) * y % MOD); }
  static int mpow(int a, int n) {
    int r = 1;
    while (n) {
      if (n % 2) r = mul(r, a);
      n /= 2, a = mul(a, a);
    }
    return r;
  }
  static int minv(int a) { return mpow(a, MOD - 2); }
  struct NTT;
  struct Poly;
  static NTT ntt;
};
template <int MOD, int RT> struct Zp<MOD, RT>::NTT {
  int w[MAXN];
  NTT() {
    int s = MAXN / 2, dw = mpow(RT, (MOD - 1) / MAXN);
    for (; s; s >>= 1, dw = mul(dw, dw)) {
      w[s] = 1;
      for (int j = 1; j < s; ++j)
        w[s + j] = mul(w[s + j - 1], dw);
    }
  }
  void apply(
    int *a, int n, bool inv = 0) { // 0 <= a_i < P
    for (int i = 0, j = 1; j < n - 1; ++j) {
      for (int k = n >> 1; (i ^= k) < k; k >>= 1);
      if (j < i) swap(a[i], a[j]);
    }
    for (int s = 1; s < n; s <<= 1) {
      for (int i = 0; i < n; i += s * 2) {
        for (int j = 0; j < s; ++j) {
          int tmp = mul(a[i + s + j], w[s + j]);
          a[i + s + j] = sub(a[i + j], tmp);
          a[i + j] = add(a[i + j], tmp);
        }
      }
    }
    if (!inv) return;
    int iv = minv(n);
    reverse(a + 1, a + n);
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], iv);
  }
};
template <int MOD, int RT>
typename Zp<MOD, RT>::NTT Zp<MOD, RT>::ntt;
using ctx1 = Zp<998244353, 3>;
int a[MAXN];
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  for (int i = 0; i < 10; ++i) {
    a[i] = rand() % 100;
    cout << a[i] << " \n"[i == 9];
  }
  ctx1::ntt.apply(a, MAXN);
  for (int i = 0; i < 10; ++i) {
    cout << a[i] << " \n"[i == 9];
  }
  ctx1::ntt.apply(a, MAXN, 1);
  for (int i = 0; i < 10; ++i) {
    cout << a[i] << " \n"[i == 9];
  }
  return 0;
}
```

7.6 Newton

Given $F(x)$ where

$$F(x)=\sum_{i=0}^{\infty}\alpha_i(x-\beta)^i$$

for β being some constant. Polynomial P such that $F(P)=0$ can be found iteratively. Denote by Q_k the polynomial such that $F(Q_k)=0 \pmod{x^{2^k}}$, then

$$Q_{k+1}=Q_k-\frac{F(Q_k)}{F'(Q_k)} \pmod{x^{2^{k+1}}}$$

7.7 Fast Fourier Transform [e5f7dc]

```
template <int MAXN> struct FFT {
  using val_t = complex<double>;
  const double PI = acos(-1);
  val_t w[MAXN];
  FFT() {
    for (int i = 0; i < MAXN; ++i) {
      double arg = 2 * PI * i / MAXN;
      w[i] = val_t(cos(arg), sin(arg));
    }
  }
  void bitrev(val_t *a, int n); // see NTT
  void trans(
    val_t *a, int n, bool inv = false); // see NTT;
  // remember to replace LL with val_t
};
```

8 Geometry

8.1 PolyUnion [bf776d]

```
double rat(pll a, pll b) {
  return sign(b.X) ? (double)a.X / b.X
                   : (double)a.Y / b.Y;
} // all poly. should be ccw
double polyUnion(vector<vector<pll>> &poly) {
  double res = 0;
  for (auto &p : poly)
    for (int a = 0; a < SZ(p); ++a) {
      pll A = p[a], B = p[(a + 1) % SZ(p)];
      vector<pair<double, int>> segs = {
        {0, 0}, {1, 0}};
      for (auto &q : poly) {
        if (&p == &q) continue;
        for (int b = 0; b < SZ(q); ++b) {
          pll C = q[b], D = q[(b + 1) % SZ(q)];
          int sc = ori(A, B, C), sd = ori(A, B, D);
          if (sc != sd && min(sc, sd) < 0) {
            double sa = cross(D - C, A - C),
                   sb = cross(D - C, B - C);
            segs.emplace_back(
              sa / (sa - sb), sign(sc - sd));
          }
          if (!sc && !sd && &q < &p &&
            sign(dot(B - A, D - C)) > 0) {
            segs.emplace_back(rat(C - A, B - A), 1);
            segs.emplace_back(rat(D - A, B - A), -1);
          }
        }
      }
      sort(ALL(segs));
      for (auto &s : segs) s.X = clamp(s.X, 0.0, 1.0);
      double sum = 0;
      int cnt = segs[0].second;
      for (int j = 1; j < SZ(segs); ++j) {
        if (!cnt) sum += segs[j].X - segs[j - 1].X;
        cnt += segs[j].Y;
      }
      res += cross(A, B) * sum;
    }
  return res / 2;
}
```

8.2 external bisector [f088cc]

```
pdd external_bisector(pdd p1, pdd p2, pdd p3) { // 213
  pdd L1 = p2 - p1, L2 = p3 - p1;
  L2 = L2 * abs(L1) / abs(L2);
  return L1 + L2;
}
```

8.3 Convexhull3D [fc330d]

```
struct convex_hull_3D {
  struct Face {
    int a, b, c;
    Face(int ta, int tb, int tc)
      : a(ta), b(tb), c(tc) {}
  }; // return the faces with pt indexes
  vector<Face> res;
  vector<Point> P;
  convex_hull_3D(const vector<Point> &_P)
    : res(), P(_P) {
    // all points coplanar case will WA, O(n^2)
    int n = SZ(P);
    if (n <= 2) return; // be careful about edge case
    // ensure first 4 points are not coplanar
    swap(P[1], *find_if(ALL(P), [&](auto p) {
      return sign(abs2(P[0] - p)) != 0;
    }));
    swap(P[2], *find_if(ALL(P), [&](auto p) {
      return sign(abs2(cross3(p, P[0], P[1]))) != 0;
    }));
    swap(P[3], *find_if(ALL(P), [&](auto p) {
      return sign(volume(P[0], P[1], P[2], p)) != 0;
    }));
    vector<vector<int>> flag(n, vector<int>(n));
    res.emplace_back(0, 1, 2);
    res.emplace_back(2, 1, 0);
    for (int i = 3; i < n; ++i) {
      vector<Face> next;
      for (auto f : res) {
        int d =
          sign(volume(P[f.a], P[f.b], P[f.c], P[i]));
        if (d <= 0) next.pb(f);
        int ff = (d > 0) - (d < 0);
        flag[f.a][f.b] = flag[f.b][f.c] =
          flag[f.c][f.a] = ff;
      }
      for (auto f : res) {
        auto F = [&](int x, int y) {
          if (flag[x][y] > 0 && flag[y][x] <= 0)
            next.emplace_back(x, y, i);
        };
        F(f.a, f.b);
        F(f.b, f.c);
        F(f.c, f.a);
      }
      res = next;
    }
  }
  bool same(Face s, Face t) {
    if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.a])) !=
      0)
      return 0;
    if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.b])) !=
      0)
      return 0;
    if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.c])) !=
      0)
      return 0;
    return 1;
  }
  int polygon_face_num() {
    int ans = 0;
    for (int i = 0; i < SZ(res); ++i)
      ans += none_of(res.begin(), res.begin() + i,
        [&](Face g) { return same(res[i], g); });
    return ans;
  }
  double get_volume() {
    double ans = 0;
    for (auto f : res)
      ans +=
        volume(Point(0, 0, 0), P[f.a], P[f.b], P[f.c]);
    return fabs(ans / 6);
  }
  double get_dis(Point p, Face f) {
    Point p1 = P[f.a], p2 = P[f.b], p3 = P[f.c];
    double a = (p2.y - p1.y) * (p3.z - p1.z) -
      (p2.z - p1.z) * (p3.y - p1.y);
    double b = (p2.z - p1.z) * (p3.x - p1.x) -
      (p2.x - p1.x) * (p3.z - p1.z);
    double c = (p2.x - p1.x) * (p3.y - p1.y) -
      (p2.y - p1.y) * (p3.x - p1.x);
    double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
    return fabs(a * p.x + b * p.y + c * p.z + d) /
      sqrt(a * a + b * b + c * c);
  }
};
// n^2 delaunay: facets with negative z normal of
// convexhull of (x, y, x^2 + y^2), use a pseudo-point
// (0, 0, inf) to avoid degenerate case
```

8.4 Triangulation Vonoroi [a4c07f]

```
// all coord. is even, you may want to call
// halfPlaneInter after then
vector<vector<Line>> vec;
void build_voronoi_line(int n, pll *arr) {
  tool.init(n, arr); // Delaunay
  vec.clear(), vec.resize(n);
  for (int i = 0; i < n; ++i)
    for (auto e : tool.head[i]) {
      int u = tool.oidx[i], v = tool.oidx[e.id];
      pll m = (arr[v] + arr[u]) / 2LL,
          d = perp(arr[v] - arr[u]);
      vec[u].pb(Line(m, m + d));
    }
}
```

8.5 Default code int [111a95]

```
typedef pair<double, double> pdd;
typedef pair<pll, pll> Line;
pll operator+(pll a, pll b) {
  return pll(a.X + b.X, a.Y + b.Y);
}
pll operator-(pll a, pll b) {
  return pll(a.X - b.X, a.Y - b.Y);
}
pll operator*(pll a, ll b) {
  return pll(a.X * b, a.Y * b);
}
pll operator/(pll a, ll b) {
  return pll(a.X / b, a.Y / b);
}
pdd operator/(pll a, double b) {
  return pdd(a.X / b, a.Y / b);
}
ll dot(pll a, pll b) { return a.X * b.X + a.Y * b.Y; }
ll cross(pll a, pll b) {
  return a.X * b.Y - a.Y * b.X;
}
ll abs2(pll a) { return dot(a, a); }
int sign(ll a) { return a == 0 ? 0 : a > 0 ? 1 : -1; }
int ori(pll a, pll b, pll c) {
  return sign(cross(b - a, c - a));
}
bool collinearity(pll p1, pll p2, pll p3) {
  return sign(cross(p1 - p3, p2 - p3)) == 0;
}
bool btw(pll p1, pll p2, pll p3) {
  if (!collinearity(p1, p2, p3)) return 0;
  return sign(dot(p1 - p3, p2 - p3)) <= 0;
}
bool seg_intersect(pll p1, pll p2, pll p3, pll p4) {
  int a123 = ori(p1, p2, p3);
  int a124 = ori(p1, p2, p4);
  int a341 = ori(p3, p4, p1);
  int a342 = ori(p3, p4, p2);
  if (a123 == 0 && a124 == 0)
    return btw(p1, p2, p3) || btw(p1, p2, p4) ||
      btw(p3, p4, p1) || btw(p3, p4, p2);
  return a123 * a124 <= 0 && a341 * a342 <= 0;
}
pdd intersect(pll p1, pll p2, pll p3, pll p4) {
  ll a123 = cross(p2 - p1, p3 - p1);
  ll a124 = cross(p2 - p1, p4 - p1);
  return (p4 * a123 - p3 * a124) /
    double(a123 - a124); // C^3 / C^2
}
pll perp(pll p1) { return pll(-p1.Y, p1.X); }
```

8.6 Polar Angle Sort [2804b5]

```
int cmp(pll a, pll b, bool same = true) {
#define is_neg(k)                                    \
  (sign(k.Y) < 0 || (sign(k.Y) == 0 && sign(k.X) < 0))
  int A = is_neg(a), B = is_neg(b);
  if (A != B) return A < B;
  if (sign(cross(a, b)) == 0)
    return same ? abs2(a) < abs2(b) : -1;
  return sign(cross(a, b)) > 0;
}
```

8.7 Default code [3efc61]

```cpp
typedef pair<double, double> pdd;
typedef pair<pdd, pdd> Line;
struct Cir {
  pdd O;
  double R;
};
const double eps = 1e-8;
pdd operator+(pdd a, pdd b) {
  return pdd(a.X + b.X, a.Y + b.Y);
}
pdd operator-(pdd a, pdd b) {
  return pdd(a.X - b.X, a.Y - b.Y);
}
pdd operator*(pdd a, double b) {
  return pdd(a.X * b, a.Y * b);
}
pdd operator/(pdd a, double b) {
  return pdd(a.X / b, a.Y / b);
}
double dot(pdd a, pdd b) {
  return a.X * b.X + a.Y * b.Y;
}
double cross(pdd a, pdd b) {
  return a.X * b.Y - a.Y * b.X;
}
double abs2(pdd a) { return dot(a, a); }
double abs(pdd a) { return sqrt(dot(a, a)); }
int sign(double a) {
  return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;
}
int ori(pdd a, pdd b, pdd c) {
  return sign(cross(b - a, c - a));
}
bool collinearity(pdd p1, pdd p2, pdd p3) {
  return sign(cross(p1 - p3, p2 - p3)) == 0;
}
bool btw(pdd p1, pdd p2, pdd p3) {
  if (!collinearity(p1, p2, p3)) return 0;
  return sign(dot(p1 - p3, p2 - p3)) <= 0;
}
bool seg_intersect(pdd p1, pdd p2, pdd p3, pdd p4) {
  int a123 = ori(p1, p2, p3);
  int a124 = ori(p1, p2, p4);
  int a341 = ori(p3, p4, p1);
  int a342 = ori(p3, p4, p2);
  if (a123 == 0 && a124 == 0)
    return btw(p1, p2, p3) || btw(p1, p2, p4) ||
      btw(p3, p4, p1) || btw(p3, p4, p2);
  return a123 * a124 <= 0 && a341 * a342 <= 0;
}
pdd intersect(pdd p1, pdd p2, pdd p3, pdd p4) {
  double a123 = cross(p2 - p1, p3 - p1);
  double a124 = cross(p2 - p1, p4 - p1);
  return (p4 * a123 - p3 * a124) /
    (a123 - a124); // C^3 / C^2
}
pdd perp(pdd p1) { return pdd(-p1.Y, p1.X); }
pdd projection(pdd p1, pdd p2, pdd p3) {
  return p1 +
    (p2 - p1) * dot(p3 - p1, p2 - p1) / abs2(p2 - p1);
}
pdd reflection(pdd p1, pdd p2, pdd p3) {
  return p3 +
    perp(p2 - p1) * cross(p3 - p1, p2 - p1) /
    abs2(p2 - p1) * 2;
}
pdd linearTransformation(
  pdd p0, pdd p1, pdd q0, pdd q1, pdd r) {
  pdd dp = p1 - p0, dq = q1 - q0,
      num(cross(dp, dq), dot(dp, dq));
  return q0 +
    pdd(cross(r - p0, num), dot(r - p0, num)) /
    abs2(dp);
} // from line p0--p1 to q0--q1, apply to r
```

8.8 PointInConvex Slow [dd78ba]

```cpp
bool PointInConvex(const vector<pll> &C, pdd p) {
  if (SZ(C) == 0) return false;
  if (SZ(C) == 1) return abs(C[0] - p) < eps;
  if (SZ(C) == 2) return btw(C[0], C[1], p);
  for (int i = 0; i < SZ(C); ++i) {
    const int j = i + 1 == SZ(C) ? 0 : i + 1;
    if (cross(C[j] - C[i], p - C[i]) < -eps)
      return false;
  }
  return true;
}
```

8.9 Intersection of polygon and circle [cbe8f5]

```cpp
// Divides into multiple triangle, and sum up
const double PI = acos(-1);
double _area(pdd pa, pdd pb, double r) {
  if (abs(pa) < abs(pb)) swap(pa, pb);
  if (abs(pb) < eps) return 0;
  double S, h, theta;
  double a = abs(pb), b = abs(pa), c = abs(pb - pa);
  double cosB = dot(pb, pb - pa) / a / c,
         B = acos(cosB);
  double cosC = dot(pa, pb) / a / b, C = acos(cosC);
  if (a > r) {
    S = (C / 2) * r * r;
    h = a * b * sin(C) / c;
    if (h < r && B < PI / 2)
      S -= (acos(h / r) * r * r -
        h * sqrt(r * r - h * h));
  } else if (b > r) {
    theta = PI - B - asin(sin(B) / r * a);
    S = .5 * a * r * sin(theta) +
      (C - theta) / 2 * r * r;
  } else S = .5 * sin(C) * a * b;
  return S;
}
double area_poly_circle(const vector<pdd> poly,
  const pdd &O, const double r) {
  double S = 0;
  for (int i = 0; i < SZ(poly); ++i)
    S += _area(poly[i] - O,
           poly[(i + 1) % SZ(poly)] - O, r) *
      ori(O, poly[i], poly[(i + 1) % SZ(poly)]);
  return fabs(S);
}
```

8.10 Tangent line of two circles [5ad86c]

```cpp
vector<Line> go(
  const Cir &c1, const Cir &c2, int sign1) {
  // sign1 = 1 for outer tang, -1 for inter tang
  vector<Line> ret;
  double d_sq = abs2(c1.O - c2.O);
  if (sign(d_sq) == 0) return ret;
  double d = sqrt(d_sq);
  pdd v = (c2.O - c1.O) / d;
  double c = (c1.R - sign1 * c2.R) / d;
  if (c * c > 1) return ret;
  double h = sqrt(max(0.0, 1.0 - c * c));
  for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
    pdd n = pdd(v.X * c - sign2 * h * v.Y,
      v.Y * c + sign2 * h * v.X);
    pdd p1 = c1.O + n * c1.R;
    pdd p2 = c2.O + n * (c2.R * sign1);
    if (sign(p1.X - p2.X) == 0 and
      sign(p1.Y - p2.Y) == 0)
      p2 = p1 + perp(c2.O - c1.O);
    ret.pb(Line(p1, p2));
  }
  return ret;
}
```

8.11 CircleCover [1d09aa]

```cpp
const int N = 1021;
struct CircleCover {
  int C;
  Cir c[N];
  bool g[N][N], overlap[N][N];
  // Area[i] : area covered by at least i circles
  double Area[N];
  void init(int _C) { C = _C; }
  struct Teve {
    pdd p;
    double ang;
    int add;
    Teve() {}
    Teve(pdd _a, double _b, int _c)
      : p(_a), ang(_b), add(_c) {}
    bool operator<(const Teve &a) const {
      return ang < a.ang;
    }
  } eve[N * 2];
  // strict: x = 0, otherwise x = -1
```

```
  bool disjuct(Cir &a, Cir &b, int x) {
    return sign(abs(a.O - b.O) - a.R - b.R) > x;
  }
  bool contain(Cir &a, Cir &b, int x) {
    return sign(a.R - b.R - abs(a.O - b.O)) > x;
  }
  bool contain(int i, int j) {
    /* c[j] is non-strictly in c[i]. */
    return (sign(c[i].R - c[j].R) > 0 ||
            (sign(c[i].R - c[j].R) == 0 && i < j)) &&
      contain(c[i], c[j], -1);
  }
  void solve() {
    fill_n(Area, C + 2, 0);
    for (int i = 0; i < C; ++i)
      for (int j = 0; j < C; ++j)
        overlap[i][j] = contain(i, j);
    for (int i = 0; i < C; ++i)
      for (int j = 0; j < C; ++j)
        g[i][j] = !(overlap[i][j] || overlap[j][i] ||
          disjuct(c[i], c[j], -1));
    for (int i = 0; i < C; ++i) {
      int E = 0, cnt = 1;
      for (int j = 0; j < C; ++j)
        if (j != i && overlap[j][i]) ++cnt;
      for (int j = 0; j < C; ++j)
        if (i != j && g[i][j]) {
          pdd aa, bb;
          CCinter(c[i], c[j], aa, bb);
          double A =
            atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
          double B =
            atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
          eve[E++] = Teve(bb, B, 1);
          eve[E++] = Teve(aa, A, -1);
          if (B > A) ++cnt;
        }
      if (E == 0) Area[cnt] += pi * c[i].R * c[i].R;
      else {
        sort(eve, eve + E);
        eve[E] = eve[0];
        for (int j = 0; j < E; ++j) {
          cnt += eve[j].add;
          Area[cnt] +=
            cross(eve[j].p, eve[j + 1].p) * .5;
          double theta = eve[j + 1].ang - eve[j].ang;
          if (theta < 0) theta += 2. * pi;
          Area[cnt] += (theta - sin(theta)) * c[i].R *
            c[i].R * .5;
        }
      }
    }
  }
};
```

8.12 Heart [4698ba]

```
pdd circenter(
  pdd p0, pdd p1, pdd p2) { // radius = abs(center)
  p1 = p1 - p0, p2 = p2 - p0;
  double x1 = p1.X, y1 = p1.Y, x2 = p2.X, y2 = p2.Y;
  double m = 2. * (x1 * y2 - y1 * x2);
  center.X = (x1 * x1 * y2 - x2 * x2 * y1 +
              y1 * y2 * (y1 - y2)) /
    m;
  center.Y = (x1 * x2 * (x2 - x1) - y1 * y1 * x2 +
              x1 * y2 * y2) /
    m;
  return center + p0;
}
pdd incenter(
  pdd p1, pdd p2, pdd p3) { // radius = area / s * 2
  double a = abs(p2 - p3), b = abs(p1 - p3),
         c = abs(p1 - p2);
  double s = a + b + c;
  return (a * p1 + b * p2 + c * p3) / s;
}
pdd masscenter(pdd p1, pdd p2, pdd p3) {
  return (p1 + p2 + p3) / 3;
}
pdd orthcenter(pdd p1, pdd p2, pdd p3) {
  return masscenter(p1, p2, p3) * 3 -
    circenter(p1, p2, p3) * 2;
}
```

8.13 PointSegDist [5ee686]

```
double PointSegDist(pdd q0, pdd q1, pdd p) {
  if (sign(abs(q0 - q1)) == 0) return abs(q0 - p);
  if (sign(dot(q1 - q0, p - q0)) >= 0 &&
    sign(dot(q0 - q1, p - q1)) >= 0)
    return fabs(cross(q1 - q0, p - q0) / abs(q0 - q1));
  return min(abs(p - q0), abs(p - q1));
}
```

8.14 Minkowski Sum [95f4a0]

```
vector<pll> Minkowski(
  vector<pll> A, vector<pll> B) { // |A|,|B|>=3
  hull(A), hull(B);
  vector<pll> C(1, A[0] + B[0]), s1, s2;
  for (int i = 0; i < SZ(A); ++i)
    s1.pb(A[(i + 1) % SZ(A)] - A[i]);
  for (int i = 0; i < SZ(B); i++)
    s2.pb(B[(i + 1) % SZ(B)] - B[i]);
  for (int i = 0, j = 0; i < SZ(A) || j < SZ(B);)
    if (j >= SZ(B) ||
      (i < SZ(A) && cross(s1[i], s2[j]) >= 0))
      C.pb(B[j % SZ(B)] + A[i++]);
    else C.pb(A[i % SZ(A)] + B[j++]);
  return hull(C), C;
}
```

8.15 TangentPointToHull [5668cc]

```
/* The point should be strictly out of hull
   return arbitrary point on the tangent line */
pii get_tangent(vector<pll> &C, pll p) {
  auto gao = [&](int s) {
    return cyc_tsearch(SZ(C), [&](int x, int y) {
      return ori(p, C[x], C[y]) == s;
    });
  };
  return pii(gao(1), gao(-1));
} // return (a, b), ori(p, C[a], C[b]) >= 0
```

8.16 Intersection of two circles [b062ba]

```
bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
  pdd o1 = a.O, o2 = b.O;
  double r1 = a.R, r2 = b.R, d2 = abs2(o1 - o2),
         d = sqrt(d2);
  if (d < max(r1, r2) - min(r1, r2) || d > r1 + r2)
    return 0;
  pdd u = (o1 + o2) * 0.5 +
    (o1 - o2) * ((r2 * r2 - r1 * r1) / (2 * d2));
  double A = sqrt((r1 + r2 + d) * (r1 - r2 + d) *
    (r1 + r2 - d) * (-r1 + r2 + d));
  pdd v =
    pdd(o1.Y - o2.Y, -o1.X + o2.X) * A / (2 * d2);
  p1 = u + v, p2 = u - v;
  return 1;
}
```

8.17 PointInConvex [9136f4]

```
bool PointInConvex(
  const vector<pll> &C, pll p, bool strict = true) {
  int a = 1, b = SZ(C) - 1, r = !strict;
  if (SZ(C) == 0) return false;
  if (SZ(C) < 3) return r && btw(C[0], C.back(), p);
  if (ori(C[0], C[a], C[b]) > 0) swap(a, b);
  if (ori(C[0], C[a], p) >= r ||
    ori(C[0], C[b], p) <= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (ori(C[0], C[c], p) > 0 ? b : a) = c;
  }
  return ori(C[a], C[b], p) < r;
}
```

8.18 Intersection of line and circle [894afd]

```
vector<pdd> circleLine(pdd c, double r, pdd a, pdd b) {
  pdd p =
    a + (b - a) * dot(c - a, b - a) / abs2(b - a);
  double s = cross(b - a, c - a),
         h2 = r * r - s * s / abs2(b - a);
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  pdd h = (b - a) / abs(b - a) * sqrt(h2);
  return {p - h, p + h};
}
```

8.19 Trapezoidalization [4e01c8]

```cpp
template <class T> struct SweepLine {
  struct cmp {
    cmp(const SweepLine &_swp) : swp(_swp) {}
    bool operator()(int a, int b) const {
      if (abs(swp.get_y(a) - swp.get_y(b)) <= swp.eps)
        return swp.slope_cmp(a, b);
      return swp.get_y(a) + swp.eps < swp.get_y(b);
    }
    const SweepLine &swp;
  } _cmp;
  T curTime, eps, curQ;
  vector<Line> base;
  multiset<int, cmp> sweep;
  multiset<pair<T, int>> event;
  vector<typename multiset<int, cmp>::iterator> its;
  vector<typename multiset<pair<T, int>>::iterator>
    eits;
  bool slope_cmp(int a, int b) const {
    assert(a != -1);
    if (b == -1) return 0;
    return sign(cross(base[a].Y - base[a].X,
                base[b].Y - base[b].X)) < 0;
  }
  T get_y(int idx) const {
    if (idx == -1) return curQ;
    Line l = base[idx];
    if (l.X.X == l.Y.X) return l.Y.Y;
    return ((curTime - l.X.X) * l.Y.Y +
              (l.Y.X - curTime) * l.X.Y) /
      (l.Y.X - l.X.X);
  }
  void insert(int idx) {
    its[idx] = sweep.insert(idx);
    if (its[idx] != sweep.begin())
      update_event(*prev(its[idx]));
    update_event(idx);
    event.emplace(base[idx].Y.X, idx + 2 * SZ(base));
  }
  void erase(int idx) {
    assert(eits[idx] == event.end());
    auto p = sweep.erase(its[idx]);
    its[idx] = sweep.end();
    if (p != sweep.begin()) update_event(*prev(p));
  }
  void update_event(int idx) {
    if (eits[idx] != event.end())
      event.erase(eits[idx]);
    eits[idx] = event.end();
    auto nxt = next(its[idx]);
    if (nxt == sweep.end() || !slope_cmp(idx, *nxt))
      return;
    auto t = intersect(base[idx].X, base[idx].Y,
      base[*nxt].X, base[*nxt].Y)
                .X;
    if (t + eps < curTime ||
      t >= min(base[idx].Y.X, base[*nxt].Y.X))
      return;
    eits[idx] = event.emplace(t, idx + SZ(base));
  }
  void swp(int idx) {
    assert(eits[idx] != event.end());
    eits[idx] = event.end();
    int nxt = *next(its[idx]);
    swap((int &)*its[idx], (int &)*its[nxt]);
    swap(its[idx], its[nxt]);
    if (its[nxt] != sweep.begin())
      update_event(*prev(its[nxt]));
    update_event(idx);
  }
  // only expected to call the functions below
  SweepLine(T t, T e, vector<Line> vec)
    : _cmp(*this), curTime(t), eps(e), curQ(),
      base(vec), sweep(_cmp), event(),
      its(SZ(vec), sweep.end()),
      eits(SZ(vec), event.end()) {
    for (int i = 0; i < SZ(base); ++i) {
      auto &[p, q] = base[i];
      if (p > q) swap(p, q);
      if (p.X <= curTime && curTime <= q.X) insert(i);
      else if (curTime < p.X) event.emplace(p.X, i);
    }
  }
  void setTime(T t, bool ers = false) {
    assert(t >= curTime);
    while (!event.empty() && event.begin()->X <= t) {
      auto [et, idx] = *event.begin();
      int s = idx / SZ(base);
      idx %= SZ(base);
      if (abs(et - t) <= eps && s == 2 && !ers) break;
      curTime = et;
      event.erase(event.begin());
      if (s == 2) erase(idx);
      else if (s == 1) swp(idx);
      else insert(idx);
    }
    curTime = t;
  }
  T nextEvent() {
    if (event.empty()) return INF;
    return event.begin()->X;
  }
  int lower_bound(T y) {
    curQ = y;
    auto p = sweep.lower_bound(-1);
    if (p == sweep.end()) return -1;
    return *p;
  }
};
```

8.20 point in circle [882728]

```cpp
// return q's relation with circumcircle of
// tri(p[0],p[1],p[2])
bool in_cc(const array<pll, 3> &p, pll q) {
  __int128 det = 0;
  for (int i = 0; i < 3; ++i)
    det += __int128(abs2(p[i]) - abs2(q)) *
      cross(p[(i + 1) % 3] - q, p[(i + 2) % 3] - q);
  return det > 0; // in: >0, on: =0, out: <0
}
```

8.21 PolyCut [417264]

```cpp
vector<pdd> cut(vector<pdd> poly, pdd s, pdd e) {
  vector<pdd> res;
  for (int i = 0; i < SZ(poly); ++i) {
    pdd cur = poly[i],
        prv = i ? poly[i - 1] : poly.back();
    bool side = ori(s, e, cur) < 0;
    if (side != (ori(s, e, prv) < 0))
      res.pb(intersect(s, e, cur, prv));
    if (side) res.pb(cur);
  }
  return res;
}
```

8.22 minDistOfTwoConvex [d62c1f]

```cpp
double ConvexHullDist(vector<pdd> A, vector<pdd> B) {
  for (auto &p : B) p = {-p.X, -p.Y};
  auto C = Minkowski(A, B); // assert SZ(C) > 0
  if (PointInConvex(C, pdd(0, 0))) return 0;
  double ans = PointSegDist(C.back(), C[0], pdd(0, 0));
  for (int i = 0; i + 1 < SZ(C); ++i) {
    ans = min(
      ans, PointSegDist(C[i], C[i + 1], pdd(0, 0)));
  }
  return ans;
}
```

8.23 DelaunayTriangulation [f2d180]

```cpp
/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle.
find : return a triangle contain given point
add_point : add a point into triangulation
A Triangle is in triangulation iff. its has_chd is 0.
Region of triangle u: iterate each u.edge[i].tri,
each points are u.p[(i+1)%3], u.p[(i+2)%3]
Voronoi diagram: for each triangle in triangulation,
the bisector of all its edges will split the region.
nearest point will belong to the triangle containing it
 */
const ll inf =
  MAXC * MAXC * 100; // lower_bound unknown
struct Tri;
struct Edge {
  Tri *tri;
  int side;
  Edge() : tri(0), side(0) {}
```

```
  Edge(Tri *_tri, int _side)
    : tri(_tri), side(_side) {}
};
struct Tri {
  pll p[3];
  Edge edge[3];
  Tri *chd[3];
  Tri() {}
  Tri(const pll &p0, const pll &p1, const pll &p2) {
    p[0] = p0;
    p[1] = p1;
    p[2] = p2;
    chd[0] = chd[1] = chd[2] = 0;
  }
  bool has_chd() const { return chd[0] != 0; }
  int num_chd() const {
    return !!chd[0] + !!chd[1] + !!chd[2];
  }
  bool contains(pll const &q) const {
    for (int i = 0; i < 3; ++i)
      if (ori(p[i], p[(i + 1) % 3], q) < 0) return 0;
    return 1;
  }
} pool[N * 10], *tris;
void edge(Edge a, Edge b) {
  if (a.tri) a.tri->edge[a.side] = b;
  if (b.tri) b.tri->edge[b.side] = a;
}
struct Trig { // Triangulation
  Trig() {
    the_root = // Tri should at least contain all
               // points
      new (tris++) Tri(pll(-inf, -inf),
        pll(inf + inf, -inf), pll(-inf, inf + inf));
  }
  Tri *find(pll p) { return find(the_root, p); }
  void add_point(const pll &p) {
    add_point(find(the_root, p), p);
  }
  Tri *the_root;
  static Tri *find(Tri *root, const pll &p) {
    while (1) {
      if (!root->has_chd()) return root;
      for (int i = 0; i < 3 && root->chd[i]; ++i)
        if (root->chd[i]->contains(p)) {
          root = root->chd[i];
          break;
        }
    }
    assert(0); // "point not found"
  }
  void add_point(Tri *root, pll const &p) {
    Tri *t[3];
    /* split it into three triangles */
    for (int i = 0; i < 3; ++i)
      t[i] = new (tris++)
        Tri(root->p[i], root->p[(i + 1) % 3], p);
    for (int i = 0; i < 3; ++i)
      edge(Edge(t[i], 0), Edge(t[(i + 1) % 3], 1));
    for (int i = 0; i < 3; ++i)
      edge(Edge(t[i], 2), root->edge[(i + 2) % 3]);
    for (int i = 0; i < 3; ++i) root->chd[i] = t[i];
    for (int i = 0; i < 3; ++i) flip(t[i], 2);
  }
  void flip(Tri *tri, int pi) {
    Tri *trj = tri->edge[pi].tri;
    int pj = tri->edge[pi].side;
    if (!trj) return;
    if (!in_cc(
          tri->p[0], tri->p[1], tri->p[2], trj->p[pj]))
      return;
    /* flip edge between tri,trj */
    Tri *trk = new (tris++) Tri(
      tri->p[(pi + 1) % 3], trj->p[pj], tri->p[pi]);
    Tri *trl = new (tris++) Tri(
      trj->p[(pj + 1) % 3], tri->p[pi], trj->p[pj]);
    edge(Edge(trk, 0), Edge(trl, 0));
    edge(Edge(trk, 1), tri->edge[(pi + 2) % 3]);
    edge(Edge(trk, 2), trj->edge[(pj + 1) % 3]);
    edge(Edge(trl, 1), trj->edge[(pj + 2) % 3]);
    edge(Edge(trl, 2), tri->edge[(pi + 1) % 3]);
    tri->chd[0] = trk;
    tri->chd[1] = trl;
    tri->chd[2] = 0;
    trj->chd[0] = trk;
    trj->chd[1] = trl;
    trj->chd[2] = 0;
    flip(trk, 1);
    flip(trk, 2);
    flip(trl, 1);
    flip(trl, 2);
  }
};
vector<Tri *> triang; // vector of all triangle
set<Tri *> vst;
void go(Tri *now) { // store all tri into triang
  if (vst.find(now) != vst.end()) return;
  vst.insert(now);
  if (!now->has_chd()) return triang.pb(now);
  for (int i = 0; i < now->num_chd(); ++i)
    go(now->chd[i]);
}
void build(int n, pll *ps) { // build triangulation
  tris = pool;
  triang.clear();
  vst.clear();
  random_shuffle(ps, ps + n);
  Trig tri; // the triangulation structure
  for (int i = 0; i < n; ++i) tri.add_point(ps[i]);
  go(tri.the_root);
}
```

8.24 rotatingSweepLine [374fec]

```
void rotatingSweepLine(vector<pii> &ps) {
  int n = SZ(ps), m = 0;
  vector<int> id(n), pos(n);
  vector<pii> line(n * (n - 1));
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      if (i != j) line[m++] = pii(i, j);
  sort(ALL(line), [&](pii a, pii b) {
    return cmp(ps[a.Y] - ps[a.X], ps[b.Y] - ps[b.X]);
  }); // cmp(): polar angle compare
  iota(ALL(id), 0);
  sort(ALL(id), [&](int a, int b) {
    if (ps[a].Y != ps[b].Y) return ps[a].Y < ps[b].Y;
    return ps[a] < ps[b];
  }); // initial order, since (1, 0) is the smallest
  for (int i = 0; i < n; ++i) pos[id[i]] = i;
  for (int i = 0; i < m; ++i) {
    auto l = line[i];
    // do something
    tie(
      pos[l.X], pos[l.Y], id[pos[l.X]], id[pos[l.Y]]) =
      make_tuple(pos[l.Y], pos[l.X], l.Y, l.X);
  }
}
```

8.25 Intersection of line and convex [e14a5c]

```
int TangentDir(vector<pll> &C, pll dir) {
  return cyc_tsearch(SZ(C), [&](int a, int b) {
    return cross(dir, C[a]) > cross(dir, C[b]);
  });
}
#define cmpL(i) sign(cross(C[i] - a, b - a))
pii lineHull(pll a, pll b, vector<pll> &C) {
  int A = TangentDir(C, a - b);
  int B = TangentDir(C, b - a);
  int n = SZ(C);
  if (cmpL(A) < 0 || cmpL(B) > 0)
    return pii(-1, -1); // no collision
  auto gao = [&](int l, int r) {
    for (int t = l; (l + 1) % n != r;) {
      int m = ((l + r + (l < r ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(t) ? l : r) = m;
    }
    return (l + !cmpL(r)) % n;
  };
  pii res = pii(gao(B, A), gao(A, B)); // (i, j)
  if (res.X == res.Y) // touching the corner i
    return pii(res.X, -1);
  if (!cmpL(res.X) &&
    !cmpL(res.Y)) // along side i, i+1
    switch ((res.X - res.Y + n + 1) % n) {
    case 0: return pii(res.X, res.X);
    case 2: return pii(res.Y, res.Y);
    }
  /* crossing sides (i, i+1) and (j, j+1)
  crossing corner i is treated as side (i, i+1)
  returned in the same order as the line hits the
  convex */
```

```
  return res;
} // convex cut: (r, l]
```

8.26 3Dpoint [90da48]

```
struct Point {
  double x, y, z;
  Point(double _x = 0, double _y = 0, double _z = 0)
    : x(_x), y(_y), z(_z) {}
  Point(pdd p) { x = p.X, y = p.Y, z = abs2(p); }
};
Point operator-(Point p1, Point p2) {
  return Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);
}
Point operator+(Point p1, Point p2) {
  return Point(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z);
}
Point operator*(Point p1, double v) {
  return Point(p1.x * v, p1.y * v, p1.z * v);
}
Point operator/(Point p1, double v) {
  return Point(p1.x / v, p1.y / v, p1.z / v);
}
Point cross(Point p1, Point p2) {
  return Point(p1.y * p2.z - p1.z * p2.y,
    p1.z * p2.x - p1.x * p2.z,
    p1.x * p2.y - p1.y * p2.x);
}
double dot(Point p1, Point p2) {
  return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}
double abs(Point a) { return sqrt(dot(a, a)); }
Point cross3(Point a, Point b, Point c) {
  return cross(b - a, c - a);
}
double area(Point a, Point b, Point c) {
  return abs(cross3(a, b, c));
}
double volume(Point a, Point b, Point c, Point d) {
  return dot(cross3(a, b, c), d - a);
}
// Azimuthal angle (longitude) to x-axis in interval
// [-pi, pi]
double phi(Point p) { return atan2(p.y, p.x); }
// Zenith angle (latitude) to the z-axis in interval
// [0, pi]
double theta(Point p) {
  return atan2(sqrt(p.x * p.x + p.y * p.y), p.z);
}
Point masscenter(Point a, Point b, Point c, Point d) {
  return (a + b + c + d) / 4;
}
pdd proj(Point a, Point b, Point c, Point u) {
  // proj. u to the plane of a, b, and c
  Point e1 = b - a;
  Point e2 = c - a;
  e1 = e1 / abs(e1);
  e2 = e2 - e1 * dot(e2, e1);
  e2 = e2 / abs(e2);
  Point p = u - a;
  return pdd(dot(p, e1), dot(p, e2));
}
Point rotate_around(
  Point p, double angle, Point axis) {
  double s = sin(angle), c = cos(angle);
  Point u = axis / abs(axis);
  return u * dot(u, p) * (1 - c) + p * c +
    cross(u, p) * s;
}
```

8.27 HPIGeneralLine [e36115]

```
using i128 = __int128;
struct LN {
  ll a, b, c; // ax + by + c <= 0
  pll dir() const { return pll(a, b); }
  LN(ll ta, ll tb, ll tc) : a(ta), b(tb), c(tc) {}
  LN(pll S, pll T)
    : a((T - S).Y), b(-(T - S).X), c(cross(T, S)) {}
};
pdd intersect(LN A, LN B) {
  double c = cross(A.dir(), B.dir());
  i128 a = i128(A.c) * B.a - i128(B.c) * A.a;
  i128 b = i128(A.c) * B.b - i128(B.c) * A.b;
  return pdd(-b / c, a / c);
}
bool cov(LN l, LN A, LN B) {
  i128 c = cross(A.dir(), B.dir());
  i128 a = i128(A.c) * B.a - i128(B.c) * A.a;
  i128 b = i128(A.c) * B.b - i128(B.c) * A.b;
  return sign(a * l.b - b * l.a + c * l.c) * sign(c) >=
    0;
}
bool operator<(LN a, LN b) {
  if (int c = cmp(a.dir(), b.dir(), false); c != -1)
    return c;
  return i128(abs(b.a) + abs(b.b)) * a.c >
    i128(abs(a.a) + abs(a.b)) * b.c;
}
```

8.28 minMaxEnclosingRectangle [d47db9]

```
const double INF = 1e18, qi = acos(-1) / 2 * 3;
pdd solve(vector<pll> &dots) {
#define diff(u, v) (dots[u] - dots[v])
#define vec(v) (dots[v] - dots[i])
  hull(dots);
  double Max = 0, Min = INF, deg;
  int n = SZ(dots);
  dots.pb(dots[0]);
  for (int i = 0, u = 1, r = 1, l = 1; i < n; ++i) {
    pll nw = vec(i + 1);
    while (cross(nw, vec(u + 1)) > cross(nw, vec(u)))
      u = (u + 1) % n;
    while (dot(nw, vec(r + 1)) > dot(nw, vec(r)))
      r = (r + 1) % n;
    if (!i) l = (r + 1) % n;
    while (dot(nw, vec(l + 1)) < dot(nw, vec(l)))
      l = (l + 1) % n;
    Min = min(Min,
      (double)(dot(nw, vec(r)) - dot(nw, vec(l))) *
        cross(nw, vec(u)) / abs2(nw));
    deg = acos(dot(diff(r, l), vec(u)) /
      abs(diff(r, l)) / abs(vec(u)));
    deg = (qi - deg) / 2;
    Max = max(Max,
      abs(diff(r, l)) * abs(vec(u)) * sin(deg) *
        sin(deg));
  }
  return pdd(Min, Max);
}
```

8.29 Half plane intersection [c3e180]

```
pll area_pair(Line a, Line b) {
  return pll(cross(a.Y - a.X, b.X - a.X),
    cross(a.Y - a.X, b.Y - a.X));
}
bool isin(Line l0, Line l1, Line l2) {
  // Check inter(l1, l2) strictly in l0
  auto [a02X, a02Y] = area_pair(l0, l2);
  auto [a12X, a12Y] = area_pair(l1, l2);
  if (a12X - a12Y < 0) a12X *= -1, a12Y *= -1;
  return (__int128)a02Y * a12X -
    (__int128)a02X * a12Y >
    0;
}
/* Having solution, check size > 2 */
/* --^-- Line.X --^-- Line.Y --^-- */
vector<Line> halfPlaneInter(vector<Line> arr) {
  sort(ALL(arr), [&](Line a, Line b) -> int {
    if (cmp(a.Y - a.X, b.Y - b.X, 0) != -1)
      return cmp(a.Y - a.X, b.Y - b.X, 0);
    return ori(a.X, a.Y, b.Y) < 0;
  });
  deque<Line> dq(1, arr[0]);
  auto pop_back = [&](int t, Line p) {
    while (SZ(dq) >= t &&
      !isin(p, dq[SZ(dq) - 2], dq.back()))
      dq.pop_back();
  };
  auto pop_front = [&](int t, Line p) {
    while (SZ(dq) >= t && !isin(p, dq[0], dq[1]))
      dq.pop_front();
  };
  for (auto p : arr)
    if (cmp(dq.back().Y - dq.back().X, p.Y - p.X, 0) !=
      -1)
      pop_back(2, p), pop_front(2, p), dq.pb(p);
  pop_back(3, dq[0]), pop_front(3, dq.back());
  return vector<Line>(ALL(dq));
}
```

8.30 Vector in poly [6d98e8]

```cpp
// ori(a, b, c) >= 0, valid: "strict" angle from a-b to
// a-c
bool btwangle(pll a, pll b, pll c, pll p, int strict) {
  return ori(a, b, p) >= strict &&
    ori(a, p, c) >= strict;
}
// whether vector{cur, p} in counter-clockwise order
// prv, cur, nxt
bool inside(
  pll prv, pll cur, pll nxt, pll p, int strict) {
  if (ori(cur, nxt, prv) >= 0)
    return btwangle(cur, nxt, prv, p, strict);
  return !btwangle(cur, prv, nxt, p, !strict);
}
```

8.31 DelaunayTriangulation dq [e6fa02]

```cpp
/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle. */
struct Edge {
  int id; // oidx[id]
  list<Edge>::iterator twin;
  Edge(int _id = 0) : id(_id) {}
};
struct Delaunay { // 0-base
  int n, oidx[N];
  list<Edge> head[N]; // result udir. graph
  pll p[N];
  void init(int _n, pll _p[]) {
    n = _n, iota(oidx, oidx + n, 0);
    for (int i = 0; i < n; ++i) head[i].clear();
    sort(oidx, oidx + n,
      [&](int a, int b) { return _p[a] < _p[b]; });
    for (int i = 0; i < n; ++i) p[i] = _p[oidx[i]];
    divide(0, n - 1);
  }
  void addEdge(int u, int v) {
    head[u].push_front(Edge(v));
    head[v].push_front(Edge(u));
    head[u].begin()->twin = head[v].begin();
    head[v].begin()->twin = head[u].begin();
  }
  void divide(int l, int r) {
    if (l == r) return;
    if (l + 1 == r) return addEdge(l, l + 1);
    int mid = (l + r) >> 1, nw[2] = {l, r};
    divide(l, mid), divide(mid + 1, r);
    auto gao = [&](int t) {
      pll pt[2] = {p[nw[0]], p[nw[1]]};
      for (auto it : head[nw[t]]) {
        int v = ori(pt[1], pt[0], p[it.id]);
        if (v > 0 ||
          (v == 0 &&
            abs2(pt[t ^ 1] - p[it.id]) <
              abs2(pt[1] - pt[0])))
          return nw[t] = it.id, true;
      }
      return false;
    };
    while (gao(0) || gao(1));
    addEdge(nw[0], nw[1]); // add tangent
    while (true) {
      pll pt[2] = {p[nw[0]], p[nw[1]]};
      int ch = -1, sd = 0;
      for (int t = 0; t < 2; ++t)
        for (auto it : head[nw[t]])
          if (ori(pt[0], pt[1], p[it.id]) > 0 &&
            (ch == -1 ||
              in_cc({pt[0], pt[1], p[ch]}, p[it.id]))){
            ch = it.id, sd = t;
      if (ch == -1) break; // upper common tangent
      for (auto it = head[nw[sd]].begin();
           it != head[nw[sd]].end();)
        if (seg_strict_intersect(
              pt[sd], p[it->id], pt[sd ^ 1], p[ch]))
          head[it->id].erase(it->twin),
            head[nw[sd]].erase(it++);
        else ++it;
      nw[sd] = ch, addEdge(nw[0], nw[1]);
    }
  }
} tool;
```

8.32 Minimum Enclosing Circle [5f3cdb]

```cpp
pdd Minimum_Enclosing_Circle(
  vector<pdd> dots, double &r) {
  pdd cent;
  random_shuffle(ALL(dots));
  cent = dots[0], r = 0;
  for (int i = 1; i < SZ(dots); ++i)
    if (abs(dots[i] - cent) > r) {
      cent = dots[i], r = 0;
      for (int j = 0; j < i; ++j)
        if (abs(dots[j] - cent) > r) {
          cent = (dots[i] + dots[j]) / 2;
          r = abs(dots[i] - cent);
          for (int k = 0; k < j; ++k)
            if (abs(dots[k] - cent) > r)
              cent =
                excenter(dots[i], dots[j], dots[k], r);
        }
    }
  return cent;
}
```

8.33 Convex hull [2a3008]

```cpp
void hull(vector<pll> &dots) { // n=1 => ans = {}
  sort(dots.begin(), dots.end());
  vector<pll> ans(1, dots[0]);
  for (int ct = 0; ct < 2; ++ct, reverse(ALL(dots)))
    for (int i = 1, t = SZ(ans); i < SZ(dots);
         ans.pb(dots[i++]))
      while (SZ(ans) > t &&
        ori(ans[SZ(ans) - 2], ans.back(), dots[i]) <=
          0)
        ans.pop_back();
  ans.pop_back(), ans.swap(dots);
}
```

9 Else

9.1 ManhattanMST [90cf5a]

```cpp
void solve(Point *a, int n) {
  sort(a, a + n, [](const Point &p, const Point &q) {
    return p.x + p.y < q.x + q.y;
  });
  set<Point> st; // greater<Point::x>
  for (int i = 0; i < n; ++i) {
    for (auto it = st.lower_bound(a[i]);
         it != st.end(); it = st.erase(it)) {
      if (it->x - it->y < a[i].x - a[i].y) break;
      es.push_back({it->u, a[i].u, dist(*it, a[i])});
    }
    st.insert(a[i]);
  }
}
void MST(Point *a, int n) {
  for (int t = 0; t < 2; ++t) {
    solve(a, n);
    for (int i = 0; i < n; ++i) swap(a[i].x, a[i].y);
    solve(a, n);
    for (int i = 0; i < n; ++i) a[i].x = -a[i].x;
  }
}
```

9.2 Mos Algorithm With modification [021725]

```cpp
/*
Mo's Algorithm With modification
Block: N^{2/3}, Complexity: N^{5/3}
*/
struct Query {
  int L, R, LBid, RBid, T;
  Query(int l, int r, int t)
    : L(l), R(r), LBid(l / blk), RBid(r / blk), T(t) {}
  bool operator<(const Query &q) const {
    if (LBid != q.LBid) return LBid < q.LBid;
    if (RBid != q.RBid) return RBid < q.RBid;
    return T < b.T;
  }
};
void solve(vector<Query> query) {
  sort(ALL(query));
  int L = 0, R = 0, T = -1;
  for (auto q : query) {
    while (T < q.T) addTime(L, R, ++T); // TODO
    while (T > q.T) subTime(L, R, T--); // TODO
    while (R < q.R) add(arr[++R]); // TODO
```

```
    while (L > q.L) add(arr[--L]); // TODO
    while (R > q.R) sub(arr[R--]); // TODO
    while (L < q.L) sub(arr[L++]); // TODO
    // answer query
  }
}
```

9.3 BitsetLCS [027ab4]

```
cin >> n >> m;
for (int i = 1, x; i <= n; ++i) cin >> x, p[x].set(i);
for (int i = 1, x; i <= m; i++) {
  cin >> x, (g = f) |= p[x];
  f.shiftLeftByOne(), f.set(0);
  ((f = g - f) ^= g) &= g;
}
cout << f.count() << '\n';
```

9.4 BinarySearchOnFraction [dec1bd]

```
struct Q {
  ll p, q;
  Q go(Q b, ll d) {
    return {p + b.p * d, q + b.q * d};
  }
};
bool pred(Q);
// returns smallest p/q in [lo, hi] such that
// pred(p/q) is true, and 0 <= p,q <= N
Q frac_bs(ll N) {
  Q lo{0, 1}, hi{1, 0};
  if (pred(lo)) return lo;
  assert(pred(hi));
  bool dir = 1, L = 1, H = 1;
  for (; L || H; dir = !dir) {
    ll len = 0, step = 1;
    for (int t = 0;
         t < 2 && (t ? step /= 2 : step *= 2);)
      if (Q mid = hi.go(lo, len + step);
          mid.p > N || mid.q > N || dir ^ pred(mid))
        t++;
      else len += step;
    swap(lo, hi = hi.go(lo, len));
    (dir ? L : H) = !!len;
  }
  return dir ? hi : lo;
}
```

9.5 SubsetSum [8fa070]

```
template <size_t S> // sum(a) < S
bitset<S> SubsetSum(const int *a, int n) {
  vector<int> c(S);
  bitset<S> dp;
  dp[0] = 1;
  for (int i = 0; i < n; ++i) ++c[a[i]];
  for (size_t i = 1; i < S; ++i) {
    while (c[i] > 2) c[i] -= 2, ++c[i * 2];
    while (c[i]--) dp |= dp << i;
  }
  return dp;
}
```

9.6 DynamicConvexTrick [477879]

```
// only works for integer coordinates!! maintain max
struct Line {
  mutable ll a, b, p;
  bool operator<(const Line &rhs) const {
    return a < rhs.a;
  }
  bool operator<(ll x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
  static const ll kInf = 1e18;
  ll Div(ll a, ll b) {
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) {
      x->p = kInf;
      return 0;
    }
    if (x->a == y->a)
      x->p = x->b > y->b ? kInf : -kInf;
    else x->p = Div(y->b - x->b, x->a - y->a);
    return x->p >= y->p;
  }
  void addline(ll a, ll b) {
    auto z = insert({a, b, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y))
      isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    auto l = *lower_bound(x);
    return l.a * x + l.b;
  }
};
```

9.7 DynamicMST [a5e63b]

```
int cnt[maxn], cost[maxn], st[maxn], ed[maxn];
pair<int, int> qr[maxn];
// qr[i].first = id of edge to be changed, qr[i].second
// = weight after operation cnt[i] = number of
// operation on edge i call solve(0, q - 1, v, 0),
// where v contains edges i such that cnt[i] == 0

void contract(int l, int r, vector<int> v,
  vector<int> &x, vector<int> &y) {
  sort(v.begin(), v.end(), [&](int i, int j) {
    if (cost[i] == cost[j]) return i < j;
    return cost[i] < cost[j];
  });
  djs.save();
  for (int i = l; i <= r; ++i)
    djs.merge(st[qr[i].first], ed[qr[i].first]);
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      x.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i)
    djs.merge(st[x[i]], ed[x[i]]);
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      y.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
}

void solve(int l, int r, vector<int> v, long long c) {
  if (l == r) {
    cost[qr[l].first] = qr[l].second;
    if (st[qr[l].first] == ed[qr[l].first]) {
      printf("%lld\n", c);
      return;
    }
    int minv = qr[l].second;
    for (int i = 0; i < (int)v.size(); ++i)
      minv = min(minv, cost[v[i]]);
    printf("%lld\n", c + minv);
    return;
  }
  int m = (l + r) >> 1;
  vector<int> lv = v, rv = v;
  vector<int> x, y;
  for (int i = m + 1; i <= r; ++i) {
    cnt[qr[i].first]--;
    if (cnt[qr[i].first] == 0)
      lv.push_back(qr[i].first);
  }
  contract(l, m, lv, x, y);
  long long lc = c, rc = c;
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    lc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(l, m, y, lc);
  djs.undo();
  x.clear(), y.clear();
  for (int i = m + 1; i <= r; ++i) cnt[qr[i].first]++;
  for (int i = l; i <= m; ++i) {
    cnt[qr[i].first]--;
    if (cnt[qr[i].first] == 0)
      rv.push_back(qr[i].first);
```

```
  }
  contract(m + 1, r, rv, x, y);
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    rc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(m + 1, r, y, rc);
  djs.undo();
  for (int i = l; i <= m; ++i) cnt[qr[i].first]++;
}
```

9.8 Matroid

Start from $S=\emptyset$. In each iteration, let

- $Y_1=\{x\notin S \mid S\cup\{x\}\in I_1\}$
- $Y_2=\{x\notin S \mid S\cup\{x\}\in I_2\}$

If there exists $x\in Y_1\cap Y_2$, insert x into S. Otherwise for each $x\in S, y\notin S$, create edges

- $x\to y$ if $S-\{x\}\cup\{y\}\in I_1$.
- $y\to x$ if $S-\{x\}\cup\{y\}\in I_2$.

Find a *shortest* path (with BFS) starting from a vertex in Y_1 and ending at a vertex in Y_2 which doesn't pass through any other vertices in Y_2, and alternate the path. The size of S will be incremented by 1 in each iteration. For the weighted case, assign weight $w(x)$ to vertex x if $x\in S$ and $-w(x)$ if $x\notin S$. Find the path with the minimum number of edges among all minimum length paths and alternate it.

9.9 cyclicLCS [9b01d1]

```
#define L 0
#define LU 1
#define U 2
const int mov[3][2] = {0, -1, -1, -1, -1, 0};
int al, bl;
char a[MAXL * 2], b[MAXL * 2]; // 0-indexed
int dp[MAXL * 2][MAXL];
char pred[MAXL * 2][MAXL];
inline int lcs_length(int r) {
  int i = r + al, j = bl, l = 0;
  while (i > r) {
    char dir = pred[i][j];
    if (dir == LU) l++;
    i += mov[dir][0];
    j += mov[dir][1];
  }
  return l;
}
inline void reroot(int r) { // r = new base row
  int i = r, j = 1;
  while (j <= bl && pred[i][j] != LU) j++;
  if (j > bl) return;
  pred[i][j] = L;
  while (i < 2 * al && j <= bl) {
    if (pred[i + 1][j] == U) {
      i++;
      pred[i][j] = L;
    } else if (j < bl && pred[i + 1][j + 1] == LU) {
      i++;
      j++;
      pred[i][j] = L;
    } else {
      j++;
    }
  }
}
int cyclic_lcs() {
  // a, b, al, bl should be properly filled
  // note: a WILL be altered in process
  //       -- concatenated after itself
  char tmp[MAXL];
  if (al > bl) {
    swap(al, bl);
    strcpy(tmp, a);
    strcpy(a, b);
    strcpy(b, tmp);
  }
  strcpy(tmp, a);
  strcat(a, tmp);
  // basic lcs
  for (int i = 0; i <= 2 * al; i++) {
    dp[i][0] = 0;
    pred[i][0] = U;
  }
  for (int j = 0; j <= bl; j++) {
    dp[0][j] = 0;
    pred[0][j] = L;
  }
  for (int i = 1; i <= 2 * al; i++) {
    for (int j = 1; j <= bl; j++) {
      if (a[i - 1] == b[j - 1])
        dp[i][j] = dp[i - 1][j - 1] + 1;
      else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      if (dp[i][j - 1] == dp[i][j]) pred[i][j] = L;
      else if (a[i - 1] == b[j - 1]) pred[i][j] = LU;
      else pred[i][j] = U;
    }
  }
  // do cyclic lcs
  int clcs = 0;
  for (int i = 0; i < al; i++) {
    clcs = max(clcs, lcs_length(i));
    reroot(i + 1);
  }
  // recover a
  a[al] = '\0';
  return clcs;
}
```

9.10 HilbertCurve [bc6dec]

```
ll hilbert(int n, int x, int y) {
  ll res = 0;
  for (int s = n / 2; s; s >>= 1) {
    int rx = (x & s) > 0;
    int ry = (y & s) > 0;
    res += s * 1ll * s * ((3 * rx) ^ ry);
    if (ry == 0) {
      if (rx == 1) x = s - 1 - x, y = s - 1 - y;
      swap(x, y);
    }
  }
  return res;
} // n = 2^k
```

9.11 Mos Algorithm On Tree [90ac22]

```
/*
Mo's Algorithm On Tree
Preprocess:
1) LCA
2) dfs with in[u] = dft++, out[u] = dft++
3) ord[in[u]] = ord[out[u]] = u
4) bitset<MAXN> inset
*/
struct Query {
  int L, R, LBid, lca;
  Query(int u, int v) {
    int c = LCA(u, v);
    if (c == u || c == v)
      q.lca = -1, q.L = out[c ^ u ^ v], q.R = out[c];
    else if (out[u] < in[v])
      q.lca = c, q.L = out[u], q.R = in[v];
    else q.lca = c, q.L = out[v], q.R = in[u];
    q.Lid = q.L / blk;
  }
  bool operator<(const Query &q) const {
    if (LBid != q.LBid) return LBid < q.LBid;
    return R < q.R;
  }
};
void flip(int x) {
  if (inset[x]) sub(arr[x]); // TODO
  else add(arr[x]); // TODO
  inset[x] = ~inset[x];
}
void solve(vector<Query> query) {
  sort(ALL(query));
  int L = 0, R = 0;
  for (auto q : query) {
    while (R < q.R) flip(ord[++R]);
    while (L > q.L) flip(ord[--L]);
    while (R > q.R) flip(ord[R--]);
    while (L < q.L) flip(ord[L++]);
    if (~q.lca) add(arr[q.lca]);
    // answer query
    if (~q.lca) sub(arr[q.lca]);
  }
}
```

9.12 AdaptiveSimpson [c048eb]

```
template <typename Func, typename d = double>
struct Simpson {
  using pdd = pair<d, d>;
```

```
  Func f;
  pdd mix(pdd l, pdd r, optional<d> fm = {}) {
    d h = (r.X - l.X) / 2, v = fm.value_or(f(l.X + h));
    return {v, h / 3 * (l.Y + 4 * v + r.Y)};
  }
  d eval(pdd l, pdd r, d fm, d eps) {
    pdd m((l.X + r.X) / 2, fm);
    d s = mix(l, r, fm).second;
    auto [flm, sl] = mix(l, m);
    auto [fmr, sr] = mix(m, r);
    d delta = sl + sr - s;
    if (abs(delta) <= 15 * eps)
      return sl + sr + delta / 15;
    return eval(l, m, flm, eps / 2) +
      eval(m, r, fmr, eps / 2);
  }
  d eval(d l, d r, d eps) {
    return eval(
      {l, f(l)}, {r, f(r)}, f((l + r) / 2), eps);
  }
  d eval2(d l, d r, d eps, int k = 997) {
    d h = (r - l) / k, s = 0;
    for (int i = 0; i < k; ++i, l += h)
      s += eval(l, l + h, eps / k);
    return s;
  }
};
template <typename Func>
Simpson<Func> make_simpson(Func f) {
  return {f};
}
```

9.13 min plus convolution [b08fbf]

```
// a is convex a[i+1]-a[i] <= a[i+2]-a[i+1]
vector<int> min_plus_convolution(
  vector<int> &a, vector<int> &b) {
  int n = SZ(a), m = SZ(b);
  vector<int> c(n + m - 1, INF);
  auto dc = [&](auto Y, int l, int r, int jl, int jr) {
    if (l > r) return;
    int mid = (l + r) / 2, from = -1, &best = c[mid];
    for (int j = jl; j <= jr; ++j)
      if (int i = mid - j; i >= 0 && i < n)
        if (best > a[i] + b[j])
          best = a[i] + b[j], from = j;
    Y(Y, l, mid - 1, jl, from),
      Y(Y, mid + 1, r, from, jr);
  };
  return dc(dc, 0, n - 1 + m - 1, 0, m - 1), c;
}
```

9.14 cyc tsearch [3dac64]

```
/* bool pred(int a, int b);
f(0) ~ f(n - 1) is a cyclic-shift U-function
return idx s.t. pred(x, idx) is false forall x*/
int cyc_tsearch(int n, auto pred) {
  if (n == 1) return 0;
  int l = 0, r = n;
  bool rv = pred(1, 0);
  while (r - l > 1) {
    int m = (l + r) / 2;
    if (pred(0, m) ? rv : pred(m, (m + 1) % n)) r = m;
    else l = m;
  }
  return pred(l, r % n) ? l : r % n;
}
```

9.15 All LCS [5548b0]

```
void all_lcs(string s, string t) { // 0-base
  vector<int> h(SZ(t));
  iota(ALL(h), 0);
  for (int a = 0; a < SZ(s); ++a) {
    int v = -1;
    for (int c = 0; c < SZ(t); ++c)
      if (s[a] == t[c] || h[c] < v) swap(h[c], v);
    // LCS(s[0, a], t[b, c]) =
    // c - b + 1 - sum([h[i] >= b] | i <= c)
    // h[i] might become -1 !!
  }
}
```

9.16 NQueens [68bc5d]

```
void solve(
  vector<int> &ret, int n) { // no sol when n=2,3
  if (n % 6 == 2) {
    for (int i = 2; i <= n; i += 2) ret.pb(i);
    ret.pb(3);
    ret.pb(1);
    for (int i = 7; i <= n; i += 2) ret.pb(i);
    ret.pb(5);
  } else if (n % 6 == 3) {
    for (int i = 4; i <= n; i += 2) ret.pb(i);
    ret.pb(2);
    for (int i = 5; i <= n; i += 2) ret.pb(i);
    ret.pb(1);
    ret.pb(3);
  } else {
    for (int i = 2; i <= n; i += 2) ret.pb(i);
    for (int i = 1; i <= n; i += 2) ret.pb(i);
  }
}
```

9.17 Mos Algorithm

- Mo's Algorithm With Addition Only
  - Sort querys same as the normal Mo's algorithm.
  - For each query $[l,r]$:
  - If $l/blk = r/blk$, brute-force.
  - If $l/blk \neq curL/blk$, initialize $curL := (l/blk+1)\cdot blk, curR := curL-1$
  - If $r > curR$, increase $curR$
  - decrease $curL$ to fit l, and then undo after answering
- Mo's Algorithm With Offline Second Time
  - Require: Changing answer $\equiv$ adding $f([l,r],r+1)$.
  - Require: $f([l,r],r+1) = f([1,r],r+1) - f([1,l),r+1)$.
  - Part1: Answer all $f([1,r],r+1)$ first.
  - Part2: Store $curR \to R$ for $curL$ (reduce the space to $O(N)$), and then answer them by the second offline algorithm.
  - Note: You must do the above symmetrically for the left boundaries.

9.18 simulated annealing [60768d]

```
double factor = 100000;
const int base = 1e9; // remember to run ~ 10 times
for (int it = 1; it <= 1000000; ++it) {
  // ans: answer, nw: current value, rnd(): mt19937
  // rnd()
  if (exp(-(nw - ans) / factor) >=
    (double)(rnd() % base) / base)
    ans = nw;
  factor *= 0.99995;
}
```

9.19 DLX [0543a9]

```
#define TRAV(i, link, start)                          \
  for (int i = link[start]; i != start; i = link[i])
template <bool E> // E: Exact, NN: num of 1s, RR: num
                  // of rows
                  struct DLX {
  int lt[NN], rg[NN], up[NN], dn[NN], rw[NN], cl[NN],
    bt[NN], s[NN], head, sz, ans;
  int rows, columns;
  bool vis[NN];
  bitset<RR> sol, cur; // not sure
  void remove(int c) {
    if (E) lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
    TRAV(i, dn, c) {
      if (E) {
        TRAV(j, rg, i)
        up[dn[j]] = up[j], dn[up[j]] = dn[j],
        --s[cl[j]];
      } else {
        lt[rg[i]] = lt[i], rg[lt[i]] = rg[i];
      }
    }
  }
  void restore(int c) {
    TRAV(i, up, c) {
      if (E) {
        TRAV(j, lt, i)
        ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
      } else {
        lt[rg[i]] = rg[lt[i]] = i;
      }
    }
    if (E) lt[rg[c]] = c, rg[lt[c]] = c;
  }
  void init(int c) {
    rows = 0, columns = c;
    for (int i = 0; i < c; ++i) {
      up[i] = dn[i] = bt[i] = i;
      lt[i] = i == 0 ? c : i - 1;
      rg[i] = i == c - 1 ? c : i + 1;
```

```
      s[i] = 0;
    }
    rg[c] = 0, lt[c] = c - 1;
    up[c] = dn[c] = -1;
    head = c, sz = c + 1;
  }
  void insert(const vector<int> &col) {
    if (col.empty()) return;
    int f = sz;
    for (int i = 0; i < (int)col.size(); ++i) {
      int c = col[i], v = sz++;
      dn[bt[c]] = v;
      up[v] = bt[c], bt[c] = v;
      rg[v] = (i + 1 == (int)col.size() ? f : v + 1);
      rw[v] = rows, cl[v] = c;
      ++s[c];
      if (i > 0) lt[v] = v - 1;
    }
    ++rows, lt[f] = sz - 1;
  }
  int h() {
    int ret = 0;
    fill_n(vis, sz, false);
    TRAV(x, rg, head) {
      if (vis[x]) continue;
      vis[x] = true, ++ret;
      TRAV(i, dn, x) TRAV(j, rg, i) vis[cl[j]] = true;
    }
    return ret;
  }
  void dfs(int dep) {
    if (dep + (E ? 0 : h()) >= ans) return;
    if (rg[head] == head)
      return sol = cur, ans = dep, void();
    if (dn[rg[head]] == rg[head]) return;
    int w = rg[head];
    TRAV(x, rg, head) if (s[x] < s[w]) w = x;
    if (E) remove(w);
    TRAV(i, dn, w) {
      if (!E) remove(i);
      TRAV(j, rg, i) remove(E ? cl[j] : j);
      cur.set(rw[i]), dfs(dep + 1), cur.reset(rw[i]);
      TRAV(j, lt, i) restore(E ? cl[j] : j);
      if (!E) restore(i);
    }
    if (E) restore(w);
  }
  int solve() {
    for (int i = 0; i < columns; ++i)
      dn[bt[i]] = i, up[i] = bt[i];
    ans = 1e9, sol.reset(), dfs(0);
    return ans;
  }
};
```

9.20 tree hash [95e839]

```
ull seed;
ull shift(ull x) {
  x ^= x << 13;
  x ^= x >> 7;
  x ^= x << 17;
  return x;
}
ull dfs(int u, int f) {
  ull sum = seed;
  for (int i : G[u])
    if (i != f) sum += shift(dfs(i, u));
  return sum;
}
```

9.21 DynamicConvexTrick bb [85e4f7]

```
// only works for integer coordinates!!

bool Flag; // 0: insert Line, 1: lower_bound x
template <class val = ll,
  class compare = less<val>> // sort lines with comp
struct DynamicConvexTrick {
  static const ll minx = 0, maxx = ll(1e9) + 5;
  static compare comp;
  struct Line {
    val a, b, l, r; // line ax + b in [l, r]
    Line(val _a, val _b, val _l = minx, val _r = maxx)
      : a(_a), b(_b), l(_l), r(_r) {}
    val operator()(val x) const { return a * x + b; }
  };
  struct cmp {
    bool operator()(const Line a, const Line b) {
      if (Flag == 0) return comp(a.a, b.a);
      return a.r < b.l;
    }
  };
  inline val idiv(val a, val b) {
    return a / b - (a % b && a < 0 ^ b < 0);
  }
  set<Line, cmp> st;
  void ins(val a, val b) {
    Flag = 0;
    Line L(a, b);
    auto it = st.lower_bound(L);
    if (it != st.begin() && it != st.end())
      if (!comp(
            (*prev(it))(it->l - 1), L(it->l - 1)) &&
        !comp((*it)(it->l), L(it->l)))
        return;
    while (it != st.end()) {
      if (it->a == L.a && !comp(it->b, L.b)) return;
      if (comp((*it)(it->r), L(it->r)))
        it = st.erase(it);
      else {
        Line M = *it;
        st.erase(it);
        L.r = max(idiv(L.b - M.b, M.a - L.a), minx);
        M.l = L.r + 1;
        it = st.insert(M).X;
        break;
      }
    }
    while (it != st.begin()) {
      auto pit = prev(it);
      if (comp((*pit)(pit->l), L(pit->l)))
        st.erase(pit);
      else {
        Line M = *pit;
        st.erase(pit);
        M.r =
          min(idiv(L.b - M.b, M.a - L.a), maxx - 1);
        L.l = M.r + 1;
        st.insert(M);
        break;
      }
    }
    st.insert(L);
  }
  val operator()(val x) {
    Flag = 1;
    auto it = st.lower_bound({0, 0, x, x});
    return (*it)(x);
  }
};

DynamicConvexTrick<> DCT;
```

10 JAVA

10.1 Big number [05dd09]

```
import java.util.Scanner;
import java.math.BigInteger;

public
class JAVA {
public
  static void main(String[] args) {
    Scanner cin = new Scanner(System.in);
    String a, b, c;
    while (cin.hasNext()) {
      a = cin.next();
      b = cin.next();
      c = cin.next();
      BigInteger ia = new BigInteger(a);
      BigInteger ic = new BigInteger(c);
      if (b.charAt(0) == '+')
        System.out.printf("%s\n", ia.add(ic));
      if (b.charAt(0) == '-')
        System.out.printf("%s\n", ia.subtract(ic));
      if (b.charAt(0) == '*')
        System.out.printf("%s\n", ia.multiply(ic));
      if (b.charAt(0) == '/')
        System.out.printf("%s\n", ia.divide(ic));
    }
  }
}
```

11 Python

11.1 misc

```
from decimal import *
setcontext(Context(prec
    =MAX_PREC, Emax=MAX_EMAX, rounding=ROUND_FLOOR))
print(Decimal(input()) * Decimal(input()))
from fractions import Fraction
Fraction
    ('3.14159').limit_denominator(10).numerator # 22
```