# Contents



# 1 Basic

## 1.1 vimrc

```
se nu rnu ai hls et ru ic is sc cul
se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
syntax on
set bg=dark
ino {<CR> {<CR>}<Esc>ko<tab>
gsettings set org.gnome.
    desktop.input-sources xkb-options "['caps:escape']"
```

## 1.2 readchar [a419b9]

```
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++;
}
```

## 1.3 Black Magic [afb343]

```
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp> // rb_tree
#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));
```

## 1.4 Pragma Optimization [0ca2dd]

```
#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)
```

## 1.5 Default Code [3ecd13]

```
#include <bits/stdc++.h>
#define fi first
#define se second
#define all(a) a.begin(),a.end()
#define SZ(a) (int)(a.size())
#define pb push_back
#define eb emplace_back
#define rep(i,a,b) for(int i=a,Z=b;i<Z;++i)
#define rep1(i,a,b) for(int i=a,Z=b;i<=Z;++i)
using namespace std;
using ll = long long;
using pii = pair<int,int>;
using vi = vector<int>;
using vvi = vector<vi>;
template<typename T, typename
    U>ostream& operator<<(ostream&os,const pair<T,U>&p)
{return os<<"("<<p.first<<", "<<p.second<<")";}
template<typename T
    > ostream& operator<<(ostream&os,const vector<T>&v)
{os<<"[";for(int i=0;i<v.size()
    ;++i)os<<v[i]<<(i==v.size()-1?"]":", ");return os;}
#ifdef D
template<typename T> void _dbg(T x){cerr<<x<<"\n";}
template<typename T,typename ...
    U> void _dbg(T x,U ...y){cerr<<x<<", ";_dbg(y...);}
#define dbg(...) do{cerr<<"\033[1;33m"<<#__VA_ARGS__
    <<" = ";_dbg(__VA_ARGS__);cerr<<"\033[0m";}while(0)
#else
#define dbg(...)
#endif
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int MOD = 998244353;

void foo(){

}

signed main(){
    ios::sync_with_stdio(0);cin.tie(0);
    int t = 1;
    // cin >> t;
    while(t--) foo();
}
```

# 2 Graph

## 2.1 BCC Vertex* [6763ed]

```
struct BCC { // 0-base
  int n, dft, nbcc;
  vi low, dfn, bln, stk, is_ap, cir;
  vvi G, bcc, nG;
  void make_bcc(int u) {
    bcc.eb(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 p) {
    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 != p)
        low[u] = min(low[u], dfn[v]);
    if (p == -1 && child < 2) is_ap[u] = 0;
    if (p == -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
};
```

## 2.2 Bridge* [4da29a]

```
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]
```

## 2.3 SCC* [7ae459]

```
struct SCC { // 0-base
  int n, dft, nscc;
  vi low, dfn, bln, instack, stk;
  vvi 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]
```

## 2.4 2SAT* [f5630a]

```
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;
  }
};
```

## 2.5 MinimumMeanCycle* [3e5d2b]

```
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;
  }
};
```

## 2.6 Virtual Tree* [1b641b]

```
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]);
}
```

## 2.7 Maximum Clique Dyn* [d50aa9]

```
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;
  }
};
```

## 2.8 Minimum Steiner Tree* [62d6fb]

```
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)
```

## 2.9 Dominator Tree* [2b8b32]

```
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]);
    }
  }
};
```

## 2.10 Minimum Arborescence* [c7338d]

```
/* 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
```

## 2.11 Vizing's theorem* [2b5b01]

```
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
```

## 2.12 Minimum Clique Cover* [879472]

```
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;
  }
};
```

## 2.13 NumberofMaximalClique* [11fa26]

```
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;
  }
};
```

# 3 Data Structure

## 3.1 Discrete Trick

```
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();
```

## 3.2 zkwseg [aae74d]

```
template<class T, T (*op)(T, T), T (*e)()>
struct segtree {
    int n, sz;
    vector<T> d;
    explicit
         segtree(int n): segtree(vector<T>(n, e())) {}
    explicit segtree(const vector<T> &v) {
        n = v.size(), sz = 1;
        while(sz < n) sz <<= 1;
        d.assign(sz*2, e());
        for(int i=0; i<n; ++i) d[sz+i] = v[i];
        for(int i=sz-1;
             i>=1; --i) d[i] = op(d[i<<1], d[i<<1|1]);
    }
    void set(int p, const T &x) {
        p += sz, d[p] = x;
        while(p>>=1) d[p] = op(d[p<<1], d[p<<1|1]);
    }
    T get(int p) const { return d[sz+p]; }
    T prod(int l, int r) const {
        T sl = e(), sr = e();
        l += sz, r += sz+1;
        while(l < r) {
            if(l&1) sl = op(sl, d[l++]);
            if(r&1) sr = op(d[--r], sr);
            l >>= 1, r >>= 1;
        }
        return op(sl, sr);
    }
};
```

## 3.3 BIT kth* [e39485]

```
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;
}
```

## 3.4 Interval Container* [c54d29]

```
/* 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);
}
```

## 3.5 Leftist Tree [e91538]

```
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;
}
```

## 3.6 Heavy light Decomposition* [b004ae]

```
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])
  }
};
```

## 3.7 Centroid Decomposition* [5a24da]

```
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;
  }
};
```

## 3.8 LiChaoST* [4a4bee]

```
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); }
};
```

## 3.9 Link cut tree* [a35b5d]

```
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;
}
```

### 3.10 KDTree [375ca2]

```
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
```

# 4 Flow/Matching

## 4.1 Bipartite Matching* [784535]

```
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();
  }
};
```

## 4.2 Kuhn Munkres* [4b3863]

```
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 MincostMaxflow* [1c78db]

```
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});
  }
};
```

## 4.4 Maximum Simple Graph Matching* [0fe1c3]

```
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
};
```

## 4.5 Maximum Weight Matching* [9ffb94]

```
#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; }
};
```

### 4.6 SW-mincut [c705f5]

```
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;
```

```cpp
    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;
```

## 4.7 BoundedFlow*(Dinic*) [4a793f]

```cpp
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;
  }
};
```

## 4.8 Gomory Hu tree* [11be99]

```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;
  }
}
```

## 4.9 Minimum Cost Circulation* [ba97cf]

```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)
```

## 4.10 Flow Models

- 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 $\infty$ (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 $\infty$ 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.

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

# 5 String

## 5.1 KMP [db6286]

```
vector<int> fail_func(const string &s) {
    int n = s.size();
    vector<int> f(n);
    for(int i=1; i<n; ++i) {
        int j = f[i-1];
        while(j > 0 && s[i] != s[j]) j = f[j-1];
        f[i] = j + (s[i] == s[j]);
    }
    return f;
}
int kmp_count(const string &s, const string &p) {
    int n = s.size(), m = p.size();
    vector<int> f = fail_func(p+"$");
    int cnt = 0, j = 0;
    for(int i=0; i<n; ++i) {
        while(j > 0 && s[i] != p[j]) j = f[j-1];
        j += (s[i] == p[j]);
        if(j == m) cnt++;
    }
    return cnt;
}
```

## 5.2 Z-value* [e4e6b3]

```
vi z_value(string s) {
    int n = s.size(), b = 0;
    vi z(n);
    for(int i=1; i<n; ++i) {
        if(z[b]+b < i) z[i] = 0;
        else z[i] = min(z[b]+b-i, z[i-b]);
        while(s[z[i]] == s[z[i]+i]) z[i]++;
        if(z[i]+i > z[b]+b) b = i;
    }
    return z;
}
```

## 5.3 Manacher* [1ad8ef]

```
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;
  }
}
```

## 5.4 SAIS* [6f26bc]

```
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;
    }
  }
};
```

## 5.5 Aho-Corasick Automatan* [794a77]

```
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 Smallest Rotation [4f469f]

```
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);
}
```

## 5.7 De Bruijn sequence* [a09470]

```
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;
```

## 5.8 Extended SAM* [64c3b7]

```
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 PalTree* [d7d2cf]

```
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;
  }
};
```

### 5.10 Main Lorentz [df1ae3]

```
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 = z_value(ru), z2 = z_value(v + '#' + u),
             z3 = z_value
                 (ru + '#' + rv), z4 = z_value(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)
```

# 6 Math

## 6.1 ax+by=gcd(only exgcd *) [7b833d]

```
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 */
```

## 6.2 Floor and Ceil [692c04]

```
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)); }
```

## 6.3 Floor Enumeration [7cbcdf]

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

## 6.4 Mod Min [9118e1]

```
// 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;
}
```

## 6.5 Gaussian integer gcd [0e7740]

```
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);
}
```

## 6.6 Miller Rabin* [06308c]

```
// 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;
}
```

## 6.7 Simultaneous Equations [a231be]

```
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;
  }
};
```

## 6.8 Pollard Rho* [cfe72f]

```
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);
  }
}
```

## 6.9 Simplex Algorithm [6b4566]

```
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;
}
```

### 6.9.1 Construction

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

$\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ 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. $\sum_{1 \le i \le n} A_{ji}x_i \ge b_j \rightarrow \sum_{1 \le i \le n} -A_{ji}x_i \le -b_j$
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 $x_i - x'_i$

## 6.10 chineseRemainder [a53b6d]

```
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;
}
```

## 6.11 Factorial without prime factor* [c324f3]

```
// 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
```

## 6.12 QuadraticResidue* [e0bf30]

```
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;
}
```

## 6.13 PiCount* [cad6d4]

```
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];
}
```

## 6.14 Discrete Log* [da27bf]

```
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;
}
```

## 6.15 Berlekamp Massey [3eb6fa]

```
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;
}
```

## 6.16 Primes

```
/* 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 Theorem

- $$\sum k^1 = -\frac{1}{12}$$
  $$\sum k^4 = \frac{1}{30}[n(n+1)(2n+1)(3n^2+3n-1)]$$
  $$\sum k^5 = \frac{1}{12}[n^2(n+1)^2(2n^2+2n-1)]$$
  $$\sum k^6 = \frac{1}{42}[n(n+1)(2n+1)(3n^4+6n^3-3n+1)]$$
- Cramer's rule
  $$\begin{matrix} ax+by=e \\ cx+dy=f \end{matrix} \Rightarrow \begin{matrix} x=\dfrac{ed-bf}{ad-bc} \\ y=\dfrac{af-ec}{ad-bc} \end{matrix}$$
- 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 $a_1 \ge \dots \ge a_n$ and $b_1,\dots,b_n$ is bigraphic if and only if $\displaystyle\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\displaystyle\sum_{i=1}^{k} a_i \le \sum_{i=1}^{n} \text{min}(b_i,k)$ holds for every $1 \le k \le n$.
- Fulkerson–Chen–Anstee theorem
  A sequence $(a_1,\ b_1),\ \dots\ ,\ (a_n,\ b_n)$ of nonnegative integer pairs with $a_1 \ \ge\ \cdots\ \ge\ a_n$ is digraphic if and only if $\displaystyle\sum_{i=1}^{n} a_i \ =\ \sum_{i=1}^{n} b_i$ and $\displaystyle\sum_{i=1}^{k} a_i \le \sum_{i=1}^{k} \text{min}(b_i,k-1) + \sum_{i=k+1}^{n} \text{min}(b_i,k)$ holds for every $1 \le k \le n$.
- 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, $\theta$: 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 $f(x_1,\dots,x_n)$ when $k$ constraints $g_i(x_1,\dots,x_n) = 0$.
  - Lagrangian function $\mathcal{L}(x_1,\ \dots\ ,\ x_n,\ \lambda_1,\ \dots\ ,\ \lambda_k) \ =\ f(x_1,\ \dots\ ,\ x_n) - \sum_{i=1}^{k} \lambda_i g_i(x_1,\dots,x_n)$.
  - 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: $\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$

| | | |
|---|---|---|
| $\frac{d}{dx}\sin^{-1}x = \frac{1}{\sqrt{1-x^2}}$ | $\frac{d}{dx}\cos^{-1}x = -\frac{1}{\sqrt{1-x^2}}$ | $\frac{d}{dx}\tan^{-1}x = \frac{1}{1+x^2}$ |
| $\frac{d}{dx}\tan x = 1+\tan^2 x$ | $\int \tan ax = -\frac{\ln\lvert\cos ax\rvert}{a}$ | |
| $\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\text{erf}(x)$ | $\int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$ | |

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

- 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

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

## 6.18 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 |

## 6.19 Euclidean Algorithms

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

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

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

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

## 6.20 General Purpose Numbers

- Bernoulli numbers
  $B_0 - 1, B_1^{\pm} = \pm\frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$
  $\displaystyle\sum_{j=0}^{m} \binom{m+1}{j} B_j = 0$, EGF is $B(x) = \frac{x}{e^x - 1} = \displaystyle\sum_{n=0}^{\infty} B_n \frac{x^n}{n!}$.
  $\displaystyle S_m(n) = \sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k^{+} n^{m+1-k}$
- 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
  $\displaystyle\prod_{n=1}^{\infty} (1-x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2}\right)$
- Catalan numbers
  $\displaystyle C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$
  $C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$
- 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. $\pi(j) \ge j$, $k$ $j$:s s.t. $\pi(j) > j$.
  $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$
  $E(n,0) = E(n,n-1) = 1$
  $E(n,k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$

## 6.21 Tips for Generating Functions

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

# 7 Polynomial

## 7.1 Fast Fourier Transform [56bdd7]

```
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
};
```

## 7.2 Number Theory Transform* [f68103]

```
//(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;
    }
  }
};
```

## 7.3 Fast Walsh Transform* [c9cdb6]

```
/* 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];
}
```

## 7.4 Polynomial Operation [105808]

```
#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);
```

```cpp
    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 = {};
```

### 7.5 Value Polynomial [96cde9]

```cpp
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];
  }
};
```

### 7.6 Newton's Method

Given $F(x)$ where

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

for $\beta$ 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

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

## 8 Geometry

### 8.1 Default Code [ca77e7]

```cpp
using ld = ll;
using pdd = pair<ld, ld>;
using Line = pair<pdd, pdd>;
struct Cir{ pdd O; ld R; };
#define X first
#define Y second
// const ld eps = 1e-7;

pdd operator+(pdd a, pdd b)
{ return {a.X + b.X, a.Y + b.Y}; }
pdd operator-(pdd a, pdd b)
{ return {a.X - b.X, a.Y - b.Y}; }
pdd operator*(ld i, pdd v)
{ return {i * v.X, i * v.Y}; }
pdd operator*(pdd v, ld i)
{ return {i * v.X, i * v.Y}; }
pdd operator/(pdd v, ld i)
{ return {v.X / i, v.Y / i}; }
ld dot(pdd a, pdd b)
{ return a.X * b.X + a.Y * b.Y; }
ld cross(pdd a, pdd b)
{ return a.X * b.Y - a.Y * b.X; }
ld abs2(pdd v)
{ return v.X * v.X + v.Y * v.Y; };
ld abs(pdd v)
{ return sqrt(abs2(v)); };
int sgn(ld v)
{ return v > 0 ? 1 : (v < 0 ? -1 : 0); }
// int sgn(
    ld v){ return v > eps ? 1 : ( v < -eps ? -1 : 0); }
```

```
int ori(pdd a, pdd b, pdd c)
{ return sgn(cross(b - a, c - a)); }
bool collinearity(pdd a, pdd b, pdd c)
{ return ori(a, b, c) == 0; }
bool btw(pdd p, pdd a, pdd b)
{ return collinearity
    (p, a, b) && sgn(dot(a - p, b - p)) <= 0; }

bool seg_intersect(pdd p1, pdd p2, pdd p3, pdd p4){
  if(btw(p1, p3, p4) || btw(p2
      , p3, p4) || btw(p3, p1, p2) || btw(p4, p1, p2))
    return true;
  return ori(p1, p2, p3) * ori(p1, p2, p4) < 0 &&
    ori(p3, p4, p1) * ori(p3, p4, p2) < 0;
}
pdd intersect(pdd p1, pdd p2, pdd p3, pdd p4){
  ld a123 = cross(p2 - p1, p3 - p1);
  ld a124 = cross(p2 - p1, p4 - p1);
  return (p4 * a123 - p3 * a124) / (a123 - a124);
}
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
```

## 8.2 PointSegDist* [57b6de]

```
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));
}
```

## 8.3 Heart [468aeb]

```
pair<pdd
    , ld> circenter(pdd a, pdd b, pdd c){ // SCOPE HASH
  pdd m1 = (a+b)/2, m2 = (b+c)/2;
  pdd cent = intersect
      (m1, m1 + perp(b-a), m2, m2 + perp(c-b));
  return {cent, abs(a-cent)};
}
pair<pdd, ld> incenter
    (pdd p1, pdd p2, pdd p3) { // radius = area / s * 2
  ld a =
      abs(p2 - p3), b = abs(p1 - p3), c = abs(p1 - p2);
  pdd cent = (a * p1 + b * p2 + c * p3) / (a + b + c);
  return {cent, abs(a-cent)};
}
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; }
```

## 8.4 point in circle [02a7ca]

```
// return q'
    s relation with circumcircle of tri(p[0],p[1],p[2])
bool inCC(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
}
```

## 8.5 Convex hull* [ca62f8]

```
vector<int> getConvexHull(vector<pdd>& pts){
  vector<int> id(SZ(pts));
  iota(iter(id), 0);
  sort(iter(id
      ), [&](int x, int y){ return pts[x] < pts[y]; });
  vector<int> hull;
  for(int tt = 0; tt < 2; tt++){
    int sz = SZ(hull);
    for(int j : id){
      pdd p = pts[j];
      while(SZ(hull) - sz >= 2 &&
          cross(pts
              [hull.back()] - pts[hull[SZ(hull) - 2]],
            p - pts[hull[SZ(hull) - 2]]) <= 0)
        hull.pop_back();
      hull.pb(j);
    }
    hull.pop_back();
    reverse(iter(id));
  }
  return hull;
}
```

## 8.6 PointInConvex* [82b81e]

```
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(p, C[0], C.back());
  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;
}
```

## 8.7 TangentPointToHull* [523bc1]

```
/* 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
```

## 8.8 Intersection of line and convex [157258]

```
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]
```

## 8.9 minMaxEnclosingRectangle* [180fb8]

```
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);
}
```

## 8.10 VectorInPoly* [c6d0fa]

```
// 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);
}
```

## 8.11 PolyUnion* [434d93]

```
ld rat(pll a, pll b) {
  return sgn(b.X) ? (ld)a.X / b.X : (ld)a.Y / b.Y;
} // all poly. should be ccw
ld polyUnion(vector<vector<pll>> &poly) {
  ld 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<ld, 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) {
            ld sa = cross(D
                 - C, A - C), sb = cross(D - C, B - C);
            segs.eb(sa / (sa - sb), sgn(sc - sd));
          }
          if (!sc && !sd &&
               &q < &p && sgn(dot(B - A, D - C)) > 0) {
            segs.eb(rat(C - A, B - A), 1);
            segs.eb(rat(D - A, B - A), -1);
          }
        }
      }
      sort(all(segs));
      for (auto &s : segs) s.X = clamp(s.X, 0.0, 1.0);
      ld 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;
}
```

## 8.12 Trapezoidalization [4d3bca]

```
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;
  }
};
```

## 8.13 Polar Angle Sort* [b20533]

```
// -1: a // b (if same), 0/1: a < b
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;
}
```

## 8.14 Half plane intersection* [e13bb6]

```
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; // C^4
}
/* 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]);
  for (auto p : arr) {
    if (cmp(
        dq.back().Y - dq.back().X, p.Y - p.X, 0) == -1)
      continue;
    while (SZ(dq
        ) >= 2 && !isin(p, dq[SZ(dq) - 2], dq.back()))
      dq.pop_back();
    while (SZ(dq) >= 2 && !isin(p, dq[0], dq[1]))
      dq.pop_front();
    dq.pb(p);
  }
  while (SZ(dq)
        >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq.back()))
    dq.pop_back();
  while (SZ(dq) >= 3 && !isin(dq.back(), dq[0], dq[1]))
    dq.pop_front();
  return vector<Line>(all(dq));
}
```

## 8.15 RotatingSweepLine [37eaad]

```
struct Event {
  pll d; int u, v;
  bool operator<(const Event &b) const {
    int ret = cmp(d, b.d, false);
    return ret == -1 ? false : ret; } // no tie-break
};
void rotatingSweepLine(const vector<pll> &p) {
  const int n = SZ(p);
  vector<Event> e; e.reserve(n * (n - 1));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n
        ; j++) // pos[i] < pos[j] when the event occurs
      if (i != j) e.pb(p[j] - p[i], i, j);
  sort(all(e));
  vector<int> ord(n), pos(n);
  iota(all(ord), 0);
  sort(all(ord), [&](int i, int j) { // initial order
      return p[i].Y != p[
          j].Y ? p[i].Y < p[j].Y : p[i].X < p[j].X; });
  for (int i = 0; i < n; i++) pos[ord[i]] = i;
  // initialize
  for (int i = 0, j = 0; i < SZ(e); i = j) {
    // do something
    vector<pii> tmp;
    for (; j < SZ(e) && !(e[i] < e[j]); j++)
      tmp.pb(pii(e[j].u, e[j].v));
    sort(all(tmp), [&](pii x, pii y){
        return pii(pos[x.fi],
            pos[x.se]) < pii(pos[y.fi], pos[y.se]); });
    for (auto [x, y] : tmp) // pos[x] + 1 == pos[y]
      tie(ord[pos[x]], ord[pos[y]], pos[x], pos[y]) =
        make_tuple
            (ord[pos[y]], ord[pos[x]], pos[y], pos[x]);
  }
}
```

## 8.16 Minimum Enclosing Circle* [2c7ccd]

```
using ld = long double;
mt19937 rng(950223);  // O(n) expected time
pair<pdd, ld> minimumEnclosingCircle(vector<pdd> &pts){
  shuffle(all(pts), rng);
  pdd c = pts[0];
  ld r = 0;
  for(int i = 1; i < SZ(pts); i++){
    if(abs(pts[i] - c) <= r) continue;
    c = pts[i]; r = 0;
    for(int j = 0; j < i; j++){
      if(abs(pts[j] - c) <= r) continue;
      c = (pts[i] + pts[j]) / 2;
      r = abs(pts[i] - c);
      for(int k = 0; k < j; k++){
        if(abs(pts[k] - c) > r)
          auto [c
              , r] = circenter(pts[i], pts[j], pts[k]);
      }
    }
  }
  return {c, r};
}
```

## 8.17 Intersection of two circles* [f7a2fe]

```
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;
}
```

## 8.18 Intersection of polygon and circle* [d4d295]

```
// 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);
}
```

## 8.19 Intersection of line and circle* [76e533]

```
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 (sgn(h2) < 0) return {};
  if (sgn(h2) == 0) return {p};
  pdd h = (b - a) / abs(b - a) * sqrt(h2);
  return {p - h, p + h};
}
```

## 8.20 Tangent line of two circles [5ac5a5]

```
vector<Line
     > CCtan(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 (sgn(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 (!sgn(p1.X - p2.X) && !sgn(p1.Y - p2.Y))
      p2 = p1 + perp(c2.O - c1.O);
    ret.pb(Line(p1, p2));
  }
  return ret;
}
```

## 8.21 Minkowski Sum* [399d43]

```
void reorder_poly(vector<pdd>& pnts){
  int mn = 0;
  for(int i = 1; i < SZ(pnts); i++)
    if(pnts[i].Y < pnts[mn].Y || (pnts[i].Y ==
         pnts[mn].Y && pnts[i].X < pnts[mn].X)) mn = i;
  rotate(pnts.begin(), pnts.begin() + mn, pnts.end());
}
vector<pdd> minkowski(vector<pdd> P, vector<pdd> Q){
  reorder_poly(P); reorder_poly(Q);
  int psz = SZ(P), qsz = SZ(Q);
  P.pb(P[0]); P.pb(P[1]); Q.pb(Q[0]); Q.pb(Q[1]);
  vector<pdd> ans; int i = 0, j = 0;
  while(i < psz || j < qsz){
    ans.pb(P[i] + Q[j]);
    int t = sgn(cross(P[i + 1]-P[i], Q[j + 1]-Q[j]));
    if(t >= 0) i++; if(t <= 0) j++;
  }
  return ans;
}
```

# 9 Else

## 9.1 Cyclic Ternary Search* [9017cc]

```
/* 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;
}
```

## 9.2 Mo's Algorithm(With modification) [f05c5b]

```
/*
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 Mo's Algorithm On Tree [8331c2]

```
/*
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]);
  }
}
```

## 9.4 Additional Mo's Algorithm Trick

- 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.

## 9.5 Hilbert Curve [1274a3]

```
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
```

## 9.6 DynamicConvexTrick* [673ffd]

```
// 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 All LCS* [78a378]

```
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 !!
  }
}
```

## 9.8 DLX* [fbcf6c]

```
#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;
  }
};
```

## 9.9 Matroid Intersection

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.

## 9.10 AdaptiveSimpson* [4074b3]

```
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}; }
```

## 9.11 Simulated Annealing [de78c6]

```
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;
}
```

## 9.12 Tree Hash* [34aae5]

```
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;
}
```

## 9.13 Binary Search On Fraction [765c5a]

```
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;
}
```

## 9.14 Min Plus Convolution* [09b5c3]

```
// 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;
}
```

## 9.15 Bitset LCS [330ab1]

```
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';
```

# 10 Python

## 10.1 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
sys.set_int_max_str_digits(int(1e6))
a, b = map(int, input().split())
a = map(int, input().split())  # array
```