Contents

1 Basic

1.1 vimrc

```
set ts=4 sw=4 nu rnu et hls mouse=a
filetype indent on
sy on
inoremap jk <Esc>
inoremap {<CR> {<CR>}<C-o>O
nnoremap J 5j
nnoremap K 5k
nnoremap <F1> :w<bar>!g++ '%' -o run -std=c++20 -DLOCAL
     -Wfatal-errors -fsanitize=address,undefined -g &&
    echo done. && time ./run<CR>
```

1.2 default

```cpp
#include <bits/stdc++.h>
using namespace std;
template<ranges::range T,
    class = enable_if_t<!is_convertible_v<T,
    string_view>>>
istream& operator>>(istream &s, T &&v) {
  for (auto &&x : v) s >> x; return s;
}
template<ranges::range T,
    class = enable_if_t<!is_convertible_v<T,
    string_view>>>
ostream& operator<<(ostream &s, T &&v) {
  for (auto &&x : v) s << x << ' '; return s;
}
#ifdef LOCAL
template<class... T> void dbg(T... x) { char e{}; ((
    cerr << e << x, e = ' '), ...); }
#define debug(x...) dbg(#x, '=', x, '\n')
#else
#define debug(...) ((void)0)
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#endif
template<class T> bool chmin(T &a, T b) { return (b < a
     and (a = b, true)); }
template<class T> bool chmax(T &a, T b) { return (a < b
     and (a = b, true)); }
template<class T> inline constexpr T inf =
    numeric_limits<T>::max() / 2;
```

1.3 judge

```bash
set -e
g++ -O3 a.cpp -o a
g++ -O3 ac.cpp -o c
g++ -O3 gen.cpp -o g

for ((i=0;;i++))
do
  echo "case $i"
  ./g > inp
  time ./a < inp > wa.out
  time ./c < inp > ac.out
  diff ac.out wa.out || break
done
```

1.4 Random

```cpp
mt19937 rng(random_device{}());
i64 rand(i64 l = -lim, i64 r = lim) {
  return uniform_int_distribution<i64>(l, r)(rng);
}
double randr(double l, double r) {
  return uniform_real_distribution<double>(l, r)(rng);
}
```

1.5 Increase stack size

```
ulimit -s
```

2 Matching and Flow

2.1 Dinic

```
template<class Cap>
struct Dinic {
  struct Edge { int v; Cap w; int rev; };
  vector<vector<Edge>> G;
  int n, S, T;
  Dinic(int n, int S, int T) : n(n), S(S), T(T), G(n)
    {}
  void add_edge(int u, int v, Cap w) {
    G[u].push_back({v, w, (int)G[v].size()});
    G[v].push_back({u, 0, (int)G[u].size() - 1});
  }
  vector<int> dep;
  bool bfs() {
    dep.assign(n, 0);
    dep[S] = 1;
    queue<int> que;
    que.push(S);
    while (!que.empty()) {
      int u = que.front(); que.pop();
      for (auto [v, w, _] : G[u])
        if (!dep[v] and w) {
          dep[v] = dep[u] + 1;
          que.push(v);
        }
    }
    return dep[T] != 0;
  }
  Cap dfs(int u, Cap in) {
    if (u == T) return in;
    Cap out = 0;
    for (auto &[v, w, rev] : G[u]) {
      if (w and dep[v] == dep[u] + 1) {
        Cap f = dfs(v, min(w, in));
        w -= f, G[v][rev].w += f;
        in -= f, out += f;
        if (!in) break;
      }
    }
    if (in) dep[u] = 0;
    return out;
  }
  Cap maxflow() {
    Cap ret = 0;
    while (bfs()) {
      ret += dfs(S, inf<Cap>);
    }
    return ret;
  }
};
```

2.2 MCMF

```
template<class Cap>
struct MCMF {
  struct Edge { int v; Cap f, w; int rev; };
  vector<vector<Edge>> G;
  int n, S, T;
  MCMF(int n, int S, int T) : n(n), S(S), T(T), G(n) {}
  void add_edge(int u, int v, Cap cap, Cap cost) {
    G[u].push_back({v, cap, cost, (int)G[v].size()});
    G[v].push_back({u, 0, -cost, (int)G[u].size() - 1})
    ;
  }
  vector<Cap> dis;
  vector<bool> vis;
  bool spfa() {
    queue<int> que;
    dis.assign(n, inf<Cap>);
    vis.assign(n, false);
    que.push(S);
    vis[S] = 1;
    dis[S] = 0;
    while (!que.empty()) {
      int u = que.front(); que.pop();
      vis[u] = 0;
      for (auto [v, f, w, _] : G[u])
        if (f and chmin(dis[v], dis[u] + w))
          if (!vis[v]) que.push(v), vis[v] = 1;
    }
    return dis[T] != inf<Cap>;
  }
  Cap dfs(int u, Cap in) {
    if (u == T) return in;
    vis[u] = 1;
    Cap out = 0;
    for (auto &[v, f, w, rev] : G[u])
      if (f and !vis[v] and dis[v] == dis[u] + w) {
        Cap x = dfs(v, min(in, f));
        in -= x, out += x;
        f -= x, G[v][rev].f += x;
        if (!in) break;
      }
    if (in) dis[u] = inf<Cap>;
    vis[u] = 0;
    return out;
  }
  pair<Cap, Cap> maxflow() {
    Cap a = 0, b = 0;
    while (spfa()) {
      Cap x = dfs(S, inf<Cap>);
      a += x;
      b += x * dis[T];
    }
    return {a, b};
  }
};
```

2.3 HopcroftKarp

```
// Complexity: O(n ^ 1.5)
// edge (u \in A) -> (v \in B) : G[u].push_back(v);
struct HK {
  vector<int> l, r, a, p;
  int ans;
  HK(int n, int m, auto &G) : l(n, -1), r(m, -1), ans{}
    {
    for (bool match = true; match; ) {
      match = false;
      queue<int> q;
      a.assign(n, -1), p.assign(n, -1);
      for (int i = 0; i < n; i++)
        if (l[i] == -1) q.push(a[i] = p[i] = i);
      while (!q.empty()) {
        int z, x = q.front(); q.pop();
        if (l[a[x]] != -1) continue;
        for (int y : G[x]) {
          if (r[y] == -1) {
            for (z = y; z != -1; ) {
              r[z] = x;
              swap(l[x], z);
              x = p[x];
            }
            match = true;
            ans++;
            break;
          } else if (p[r[y]] == -1) {
            q.push(z = r[y]);
            p[z] = x;
            a[z] = a[x];
          }
        }
      }
    }
  }
};
```

2.4 KM

```
i64 KM(vector<vector<int>> W) {
  const int n = W.size();
  vector<int> fl(n, -1), fr(n, -1), hr(n), hl(n);
  for (int i = 0; i < n; ++i) {
    hl[i] = *max_element(W[i].begin(), W[i].end());
  }
  auto Bfs = [&](int s) {
    vector<int> slk(n, INF), pre(n);
    vector<bool> vl(n, false), vr(n, false);
    queue<int> que;
    que.push(s);
    vr[s] = true;
    auto Check = [&](int x) -> bool {
      if (vl[x] = true, fl[x] != -1) {
        que.push(fl[x]);
        return vr[fl[x]] = true;
```

```
      }
      while (x != -1) swap(x, fr[fl[x] = pre[x]]);
      return false;
    };
    while (true) {
      while (!que.empty()) {
        int y = que.front(); que.pop();
        for (int x = 0, d = 0; x < n; ++x) {
          if (!vl[x] and slk[x] >= (d = hl[x] + hr[y] -
    W[x][y])) {
            if (pre[x] = y, d) slk[x] = d;
            else if (!Check(x)) return;
          }
        }
      }
      int d = INF;
      for (int x = 0; x < n; ++x) {
        if (!vl[x] and 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] and !slk[x] and !Check(x)) return;
      }
    }
  };
  for (int i = 0; i < n; ++i) Bfs(i);
  i64 res = 0;
  for (int i = 0; i < n; ++i) res += W[i][fl[i]];
  return res;
}
```

2.5 SW

```
int w[kN][kN], g[kN], del[kN], v[kN];
void AddEdge(int x, int y, int c) {
  w[x][y] += c;
  w[y][x] += c;
}
pair<int, int> Phase(int n) {
  fill(v, v + n, 0), fill(g, g + n, 0);
  int s = -1, t = -1;
  while (true) {
    int c = -1;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      if (c == -1 || g[i] > g[c]) c = i;
    }
    if (c == -1) break;
    v[c] = 1, s = t, t = c;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      g[i] += w[c][i];
    }
  }
  return make_pair(s, t);
}
int GlobalMinCut(int n) {
  int cut = kInf;
  fill(del, 0, sizeof(del));
  for (int i = 0; i < n - 1; ++i) {
    int s, t; tie(s, t) = Phase(n);
    del[t] = 1, cut = min(cut, g[t]);
    for (int j = 0; j < n; ++j) {
      w[s][j] += w[t][j];
      w[j][s] += w[j][t];
    }
  }
  return cut;
}
```

2.6 GeneralMatching

```
struct GeneralMatching { // n <= 500
  const int BLOCK = 10;
  int n;
  vector<vector<int> > g;
  vector<int> hit, mat;
  std::priority_queue<pair<i64, int>, vector<pair<i64,
    int>>, greater<pair<i64, int>>> unmat;
  GeneralMatching(int _n) : n(_n), g(_n), mat(n, -1),
    hit(n) {}
  void add_edge(int a, int b) { // 0 <= a != b < n
    g[a].push_back(b);
    g[b].push_back(a);
  }
  int get_match() {
    for (int i = 0; i < n; i++) if (!g[i].empty()) {
      unmat.emplace(0, i);
    }
    // If WA, increase this
    // there are some cases that need >=1.3*n^2 steps
 for BLOCK=1
    // no idea what the actual bound needed here is.
    const int MAX_STEPS = 10 + 2 * n + n * n / BLOCK /
    2;
    mt19937 rng(random_device{}());
    for (int i = 0; i < MAX_STEPS; ++i) {
      if (unmat.empty()) break;
      int u = unmat.top().second;
      unmat.pop();
      if (mat[u] != -1) continue;
      for (int j = 0; j < BLOCK; j++) {
        ++hit[u];
        auto &e = g[u];
        const int v = e[rng() % e.size()];
        mat[u] = v;
        swap(u, mat[v]);
        if (u == -1) break;
      }
      if (u != -1) {
        mat[u] = -1;
        unmat.emplace(hit[u] * 100ULL / (g[u].size() +
 1), u);
      }
    }
    int siz = 0;
    for (auto e : mat) siz += (e != -1);
    return siz / 2;
  }
};
```

3 Graph

3.1 Strongly Connected Component

```
struct SCC {
  int n;
  vector<vector<int>> G;
  vector<int> dfn, low, id, stk;
  int scc{}, _t{};
  SCC(int _n) : n{_n}, G(_n) {}
  void dfs(int u) {
    dfn[u] = low[u] = _t++;
    stk.push_back(u);
    for (int v : G[u]) {
      if (dfn[v] == -1) {
        dfs(v);
        chmin(low[u], low[v]);
      } else if (id[v] == -1) {
        chmin(low[u], dfn[v]);
      }
    }
    if (dfn[u] == low[u]) {
      int t;
      do {
        t = stk.back();
        stk.pop_back();
        id[t] = scc;
      } while (t != u);
      scc++;
    }
  }
  void work() {
    dfn.assign(n, -1);
    low.assign(n, -1);
    id.assign(n, -1);
    for (int i = 0; i < n; i++)
      if (dfn[i] == -1) {
        dfs(i);
      }
  }
};
```

3.2 2-SAT

```
struct TwoSat {
  int n;
  vector<vector<int>> e;
  vector<bool> ans;
  TwoSat(int n) : n(n), e(2 * n), ans(n) {}
  void addClause(int u, bool f, int v, bool g) { // (u
    = f) or (v = g)
    e[2 * u + !f].push_back(2 * v + g);
    e[2 * v + !g].push_back(2 * u + f);
  }
  void addImply(int u, bool f, int v, bool g) { // (u =
     f) -> (v = g)
    e[2 * u + f].push_back(2 * v + g);
    e[2 * v + !g].push_back(2 * u + !f);
  }
  bool satisfiable() {
    vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 *
    n, -1);
    vector<int> stk;
    int now = 0, cnt = 0;
    function<void(int)> tarjan = [&](int u) {
      stk.push_back(u);
      dfn[u] = low[u] = now++;
      for (auto v : e[u]) {
        if (dfn[v] == -1) {
          tarjan(v);
          low[u] = min(low[u], low[v]);
        } else if (id[v] == -1) {
          low[u] = min(low[u], dfn[v]);
        }
      }
      if (dfn[u] == low[u]) {
        int v;
        do {
          v = stk.back();
          stk.pop_back();
          id[v] = cnt;
        } while (v != u);
        ++cnt;
      }
    };
    for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1)
    tarjan(i);
    for (int i = 0; i < n; ++i) {
      if (id[2 * i] == id[2 * i + 1]) return false;
      ans[i] = id[2 * i] > id[2 * i + 1];
    }
    return true;
  }
};
```

3.3 Tree

```
struct Tree {
  int n, lgN;
  vector<vector<int>> G, st;
  vector<int> in, out, dep, pa, seq;
  Tree(int n) : n(n), G(n), in(n), out(n), dep(n), pa(n
    , -1) {}
  int cmp(int a, int b) {
    return dep[a] < dep[b] ? a : b;
  }
  void dfs(int u) {
    in[u] = seq.size();
    seq.push_back(u);
    for (int v : G[u]) if (v != pa[u]) {
      dep[v] = dep[u] + 1;
      pa[v] = u;
      dfs(v);
    }
    out[u] = seq.size();
  }
  void build() {
    seq.reserve(n);
    dfs(0);
    lgN = __lg(n);
    st.assign(lgN + 1, vector<int>(n));
    st[0] = seq;
    for (int i = 0; i < lgN; i++)
      for (int j = 0; j + (2 << i) <= n; j++)
        st[i + 1][j] = cmp(st[i][j], st[i][j + (1 << i)
    ]);
  }
  int inside(int x, int y) {
    return in[x] <= in[y] and in[y] < out[x];
  }
  int lca(int x, int y) {
    if (x == y) return x;
    if ((x = in[x] + 1) > (y = in[y] + 1))
      swap(x, y);
    int h = __lg(y - x);
    return pa[cmp(st[h][x], st[h][y - (1 << h)])];
  }
  int dist(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[lca(x, y)];
  }
  vector<int> virTree(vector<int> ver) {
    sort(all(ver), [&](int a, int b) {
      return in[a] < in[b];
    });
    for (int i = ver.size() - 1; i > 0; i--)
      ver.push_back(lca(ver[i], ver[i - 1]));
    sort(all(ver), [&](int a, int b) {
      return in[a] < in[b];
    });
    ver.erase(unique(all(ver)), ver.end());
    return ver;
  }
  void inplace_virTree(vector<int> &ver) { // O(n),
    need sort before
    vector<int> ex;
    for (int i = 0; i + 1 < ver.size(); i++)
      if (!inside(ver[i], ver[i + 1]))
        ex.push_back(lca(ver[i], ver[i + 1]));
    vector<int> stk, pa(ex.size(), -1);
    for (int i = 0; i < ex.size(); i++) {
      int lst = -1;
      while (stk.size() and in[ex[stk.back()]] >= in[ex
    [i]]) {
        lst = stk.back();
        stk.pop_back();
      }
      if (lst != -1) pa[lst] = i;
      if (stk.size()) pa[i] = stk.back();
      stk.push_back(i);
    }
    vector<bool> vis(ex.size());
    auto dfs = [&](auto self, int u) -> void {
      vis[u] = 1;
      if (pa[u] != -1 and !vis[pa[u]])
        self(self, pa[u]);
      if (ex[u] != ver.back())
        ver.push_back(ex[u]);
    };
    const int s = ver.size();
    for (int i = 0; i < ex.size(); i++)
      if (!vis[i]) dfs(dfs, i);
    inplace_merge(ver.begin(), ver.begin() + s, ver.end
    (),
        [&](int a, int b) { return in[a] < in[b]; });
    ver.erase(unique(all(ver)), ver.end());
  }
};
```

3.4 Manhattan MST

```
vector<tuple<int, int, int>> ManhattanMST(vector<Pt> P)
    {
  vector<int> id(P.size());
  iota(all(id), 0);
  vector<tuple<int, int, int>> edges;
  for (int k = 0; k < 4; ++k) {
    sort(all(id), [&](int i, int j) -> bool {
      return (P[i] - P[j]).ff < (P[j] - P[i]).ss;
    });
    map<int, int> sweep;
    for (int i : id) {
      for (auto it = sweep.lower_bound(-P[i].ss); \
          it != sweep.end(); sweep.erase(it++)) {
        int j = it->ss;
        Pt d = P[i] - P[j];
        if (d.ss > d.ff) break;
        edges.emplace_back(d.ss + d.ff, i, j);
      }
      sweep[-P[i].ss] = i;
```

```
    }
    for (Pt &p : P) {
      if (k % 2) p.ff = -p.ff;
      else swap(p.ff, p.ss);
    }
  }
  return edges;
}
```

3.5 TreeHash

```
map<vector<int>, int> id;
vector<vector<int>> sub;
vector<int> siz;
int getid(const vector<int> &T) {
  if (id.count(T)) return id[T];
  int s = 1;
  for (int x : T) {
    s += siz[x];
  }
  sub.push_back(T);
  siz.push_back(s);
  return id[T] = id.size();
}
int dfs(int u, int f) {
  vector<int> S;
  for (int v : G[u]) if (v != f) {
    S.push_back(dfs(v, u));
  }
  sort(all(S));
  return getid(S);
}
```

3.6 Maximum IndependentSet

```
// n <= 40, (*500)
set<int> MI(const vector<vector<int>> &adj) {
  set<int> I, V;
  for (int i = 0; i < adj.size(); i++)
    V.insert(i);
  while (!V.empty()) {
    auto it = next(V.begin(), rng() % V.size());
    int cho = *it;
    I.insert(cho);
    V.extract(cho);
    for (int i : adj[cho]) {
      if (auto j = V.find(i); j != V.end())
        V.erase(j);
    }
  }
  return I;
}
```

3.7 Min Mean Weight Cycle

```
// d[i][j] == 0 if {i,j} !in E
long long d[1003][1003], dp[1003][1003];

pair<long long, long long> MMWC() {
 memset(dp, 0x3f, sizeof(dp));
 for (int i = 1; i <= n; ++i) dp[0][i] = 0;
 for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= n; ++j) {
   for (int k = 1; k <= n; ++k) {
    dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
   }
  }
 }
 long long au = 1ll << 31, ad = 1;
 for (int i = 1; i <= n; ++i) {
  if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
  long long u = 0, d = 1;
  for (int j = n - 1; j >= 0; --j) {
   if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
    u = dp[n][i] - dp[j][i];
    d = n - j;
   }
  }
  if (u * ad < au * d) au = u, ad = d;
 }
 long long g = __gcd(au, ad);
 return make_pair(au / g, ad / g);
}
```

3.8 Block Cut Tree

```
struct BlockCutTree {
  int n;
  vector<vector<int>> adj;
  BlockCutTree(int _n) : n(_n), adj(_n) {}
  void addEdge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  pair<int, vector<pair<int, int>>> work() {
    vector<int> dfn(n, -1), low(n), stk;
    vector<pair<int, int>> edg;
    int cnt = 0, cur = 0;
    function<void(int)> dfs = [&](int x) {
      stk.push_back(x);
      dfn[x] = low[x] = cur++;
      for (auto y : adj[x]) {
        if (dfn[y] == -1) {
          dfs(y);
          low[x] = min(low[x], low[y]);
          if (low[y] == dfn[x]) {
            int v;
            do {
              v = stk.back();
              stk.pop_back();
              edg.emplace_back(n + cnt, v);
            } while (v != y);
            edg.emplace_back(x, n + cnt);
            cnt++;
          }
        } else {
          low[x] = min(low[x], dfn[y]);
        }
      }
    };
    for (int i = 0; i < n; i++) {
      if (dfn[i] == -1) {
        stk.clear();
        dfs(i);
      }
    }
    return {cnt, edg};
  }
};
```

3.9 Heavy Light Decomposition

```
struct HLD {
  int n;
  vector<int> siz, top, dep, pa, in, out, seq;
  vector<vector<int>> G;
  HLD(int n) : n(n), G(n), siz(n), top(n),
    dep(n), pa(n), in(n), out(n), seq(n) {}
  int cur{};
  void addEdge(int u, int v) {
    G[u].push_back(v);
    G[v].push_back(u);
  }
  void work(int root = 0) {
    cur = 0;
    top[root] = root;
    dep[root] = 0;
    pa[root] = -1;
    dfs1(root);
    dfs2(root);
  }
  void dfs1(int u) {
    if (pa[u] != -1) {
      G[u].erase(find(all(G[u]), pa[u]));
    }
    siz[u] = 1;
    for (auto &v : G[u]) {
      pa[v] = u;
      dep[v] = dep[u] + 1;
      dfs1(v);
      siz[u] += siz[v];
      if (siz[v] > siz[G[u][0]]) {
        swap(v, G[u][0]);
      }
    }
  }
  void dfs2(int u) {
```

```cpp
    in[u] = cur++;
    seq[in[u]] = u;
    for (int v : G[u]) {
      top[v] = (v == G[u][0] ? top[u] : v);
      dfs2(v);
    }
    out[u] = cur;
  }
  int lca(int x, int y) {
    while (top[x] != top[y]) {
      if (dep[top[x]] < dep[top[y]]) swap(x, y);
      x = pa[top[x]];
    }
    return dep[x] < dep[y] ? x : y;
  }
  int dist(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[lca(x, y)];
  }
  int jump(int x, int k) {
    if (dep[x] < k) return -1;
    int d = dep[x] - k;
    while (dep[top[x]] > d) {
      x = pa[top[x]];
    }
    return seq[in[x] - dep[x] + d];
  }
  bool isAnc(int x, int y) {
    return in[x] <= in[y] and in[y] < out[x];
  }
  int rootPar(int r, int x) {
    if (r == x) return r;
    if (!isAnc(x, r)) return pa[x];
    auto it = upper_bound(all(G[x]), r, [&](int a, int b) -> bool {
      return in[a] < in[b];
    }) - 1;
    return *it;
  }
  int rootSiz(int r, int x) {
    if (r == x) return n;
    if (!isAnc(x, r)) return siz[x];
    return n - siz[rootPar(r, x)];
  }
  int rootLca(int a, int b, int c) {
    return lca(a, b) ^ lca(b, c) ^ lca(c, a);
  }
};
```

3.10 Dominator Tree

```cpp
struct Dominator {
  vector<vector<int>> g, r, rdom; int tk;
  vector<int> dfn, rev, fa, sdom, dom, val, rp;
  int n;
  Dominator(int n) : n(n), g(n), r(n), rdom(n), tk(0),
    dfn(n, -1), rev(n, -1), fa(n, -1), sdom(n, -1),
    dom(n, -1), val(n, -1), rp(n, -1) {}
  void add_edge(int x, int y) { g[x].push_back(y); }
  void dfs(int x) {
    rev[dfn[x] = tk] = x;
    fa[tk] = sdom[tk] = val[tk] = tk; tk++;
    for (int u : g[x]) {
      if (dfn[u] == -1) dfs(u), rp[dfn[u]] = dfn[x];
      r[dfn[u]].push_back(dfn[x]);
    }
  }
  void merge(int x, int y) { fa[x] = y; }
  int find(int x, int c = 0) {
    if (fa[x] == x) return c ? -1 : x;
    if (int p = find(fa[x], 1); p != -1) {
      if (sdom[val[x]] > sdom[val[fa[x]]])
        val[x] = val[fa[x]];
      fa[x] = p;
      return c ? p : val[x];
    }
    return c ? fa[x] : val[x];
  }
  vector<int> build(int s) {
    // return the father of each node in dominator tree
    // p[i] = -2 if i is unreachable from s
    dfs(s);
    for (int i = tk - 1; i >= 0; --i) {
      for (int u : r[i])
        sdom[i] = min(sdom[i], sdom[find(u)]);
      if (i) rdom[sdom[i]].push_back(i);
      for (int u : rdom[i]) {
        int p = find(u);
        dom[u] = (sdom[p] == i ? i : p);
      }
      if (i) merge(i, rp[i]);
    }
    vector<int> p(n, -2); p[s] = -1;
    for (int i = 1; i < tk; ++i)
      if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
    for (int i = 1; i < tk; ++i)
      p[rev[i]] = rev[dom[i]];
    return p;
  }
};
```

4 Data Structure

4.1 Lazy Segtree

```cpp
template<class S, class T>
struct Seg {
  Seg<S, T> *ls{}, *rs{};
  int l, r;
  S d{};
  T f{};
  Seg(int _l, int _r) : l{_l}, r{_r} {
    if (r - l == 1) {
      return;
    }
    int mid = (l + r) / 2;
    ls = new Seg(l, mid);
    rs = new Seg(mid, r);
    pull();
  }
  void upd(const T &g) {
    g(d), g(f);
  }
  void pull() {
    d = ls->d + rs->d;
  }
  void push() {
    ls->upd(f);
    rs->upd(f);
    f = T{};
  }
  S query(int x, int y) {
    if (y <= l or r <= x) return S{};
    if (x <= l and r <= y) return d;
    push();
    return ls->query(x, y) + rs->query(x, y);
  }
  void apply(int x, int y, const T &g) {
    if (y <= l or r <= x) return;
    if (x <= l and r <= y) {
      upd(g);
      return;
    }
    push();
    ls->apply(x, y, g);
    rs->apply(x, y, g);
    pull();
  }
  void set(int p, const S &g) {
    if (p + 1 <= l or r <= p) return;
    if (r - l == 1) {
      d = g;
      return;
    }
    push();
    ls->set(p, g);
    rs->set(p, g);
    pull();
  }
  int findFirst(int x, int y, auto pred) {
    if (y <= l or r <= x or !pred(d)) return -1;
    if (r - l == 1) return l;
    push();
    int res = ls->findFirst(x, y, pred);
    return res == -1 ? rs->findFirst(x, y, pred) : res;
  }
  int findLast(int x, int y, auto pred) {
```

```
    if (y <= l or r <= x or !pred(d)) return -1;
    if (r - l == 1) return l;
    push();
    int res = rs->findLast(x, y, pred);
    return res == -1 ? ls->findLast(x, y, pred) : res;
  }
};
```

4.2 Sparse Table

```
template<class T, auto F>
struct SparseTable {
  int n, lgN;
  vector<vector<T>> st;
  SparseTable(const vector<T> &V) {
    n = V.size();
    lgN = __lg(n);
    st.assign(lgN + 1, vector<T>(n));
    st[0] = V;
    for (int i = 0; (2 << i) <= n; i++)
      for (int j = 0; j + (2 << i) <= n; j++) {
        st[i + 1][j] = F(st[i][j], st[i][j + (1 << i)])
    ;
      }
  }
  T qry(int l, int r) { // [l, r)
    int h = __lg(r - l);
    return F(st[h][l], st[h][r - (1 << h)]);
  }
};
```

4.3 Binary Index Tree

```
template<class T>
struct BIT {
  int n;
  vector<T> a;
  BIT(int n) : n(n), a(n) {}
  int lowbit(int x) { return x & -x; }
  void add(int p, T x) {
    for (int i = p + 1; i <= n; i += lowbit(i))
      a[i - 1] += x;
  }
  T qry(int p) {
    T r{};
    for (int i = p + 1; i > 0; i -= lowbit(i))
      r += a[i - 1];
    return r;
  }
  T qry(int l, int r) { // [l, r)
    return qry(r - 1) - qry(l - 1);
  }
  int kth(T k) {
    int x = 0;
    for (int i = 1 << __lg(n); i; i >>= 1) {
      if (x + i <= n and k >= a[x + i - 1]) {
        x += i;
        k -= a[x - 1];
      }
    }
    return x;
  }
};
```

4.4 Special Segtree

```
struct Seg {
  Seg *ls, *rs;
  int l, r;
  vector<int> f, g;
  // f : intervals where covering [l, r]
  // g : intervals where interset with [l, r]
  Seg(int _l, int _r) : l{_l}, r{_r} {
    int mid = (l + r) >> 1;
    if (r - l == 1) return;
    ls = new Seg(l, mid);
    rs = new Seg(mid, r);
  }
  void insert(int x, int y, int id) {
    if (y <= l or r <= x) return;
    g.push_back(id);
    if (x <= l and r <= y) {
      f.push_back(id);
      return;
    }
    ls->insert(x, y, id);
    rs->insert(x, y, id);
  }
  void fix() {
    while (!f.empty() and use[f.back()]) f.pop_back();
    while (!g.empty() and use[g.back()]) g.pop_back();
  }
  int query(int x, int y) {
    if (y <= l or r <= x) return -1;
    fix();
    if (x <= l and r <= y) {
      return g.empty() ? -1 : g.back();
    }
    return max({f.empty() ? -1 : f.back(), ls->query(x,
     y), rs->query(x, y)});
  }
};
```

4.5 Treap

```
mt19937 rng(random_device{}());
template<class S, class T>
struct Treap {
  struct Node {
    Node *ls{}, *rs{};
    int pos, siz;
    u32 pri;
    S d{}, e{};
    T f{};
    Node(int p, S x) : d{x}, e{x}, pos{p}, siz{1}, pri{
    rng()} {}
    void upd(T &g) {
      g(d), g(e), g(f);
    }
    void pull() {
      siz = Siz(ls) + Siz(rs);
      d = Get(ls) + e + Get(rs);
    }
    void push() {
      if (ls) ls->upd(f);
      if (rs) rs->upd(f);
      f = T{};
    }
  } *root{};
  static int Siz(Node *p) { return p ? p->siz : 0; }
  static S Get(Node *p) { return p ? p->d : S{}; }
  Treap() : root{} {}
  Node* Merge(Node *a, Node *b) {
    if (!a or !b) return a ? a : b;
    if (a->pri < b->pri) {
      a->push();
      a->rs = Merge(a->rs, b);
      a->pull();
      return a;
    } else {
      b->push();
      b->ls = Merge(a, b->ls);
      b->pull();
      return b;
    }
  }
  void Split(Node *p, Node *&a, Node *&b, int k) {
    if (!p) return void(a = b = nullptr);
    p->push();
    if (p->pos <= k) {
      a = p;
      Split(p->rs, a->rs, b, k);
      a->pull();
    } else {
      b = p;
      Split(p->ls, a, b->ls, k);
      b->pull();
    }
  }
  void insert(int p, S x) {
    Node *L, *R;
    Split(root, L, R, p);
    root = Merge(Merge(L, new Node(p, x)), R);
  }
  void erase(int x) {
    Node *L, *M, *R;
    Split(root, M, R, x);
```

```cpp
    Split(M, L, M, x - 1);
    if (M) M = Merge(M->ls, M->rs);
    root = Merge(Merge(L, M), R);
  }
  S query() {
    return Get(root);
  }
};
```

4.6 LiChao Segtree

```cpp
struct Line {
  i64 k, m; // y = k + mx;
  Line() : k{INF}, m{} {}
  Line(i64 _k, i64 _m) : k(_k), m(_m) {}
  i64 get(i64 x) {
    return k + m * x;
  }
};
struct Seg {
  Seg *ls{}, *rs{};
  int l, r, mid;
  Line line{};
  Seg(int _l, int _r) : l(_l), r(_r), mid(_l + _r >> 1)
     {
    if (r - l == 1) return;
    ls = new Seg(l, mid);
    rs = new Seg(mid, r);
  }
  void insert(Line L) {
    if (line.get(mid) > L.get(mid))
      swap(line, L);
    if (r - l == 1) return;
    if (L.m < line.m) {
      rs->insert(L);
    } else {
      ls->insert(L);
    }
  }
  i64 query(int p) {
    if (p < l or r <= p) return INF;
    if (r - l == 1) return line.get(p);
    return min({line.get(p), ls->query(p), rs->query(p)
    });
  }
};
```

4.7 Persistent SegmentTree

```cpp
template<class S>
struct Seg {
  Seg *ls{}, *rs{};
  int l, r;
  S d{};
  Seg(Seg* p) { (*this) = *p; }
  Seg(int l, int r) : l(l), r(r) {
    if (r - l == 1) {
      d = {};
      return;
    }
    int mid = (l + r) / 2;
    ls = new Seg(l, mid);
    rs = new Seg(mid, r);
    pull();
  }
  void pull() {
    d = ls->d + rs->d;
  }
  Seg* set(int p, const S &x) {
    Seg* n = new Seg(this);
    if (r - l == 1) {
      n->d = x;
      return n;
    }
    int mid = (l + r) / 2;
    if (p < mid) {
      n->ls = ls->set(p, x);
    } else {
      n->rs = rs->set(p, x);
    }
    n->pull();
    return n;
  }
  S query(int x, int y) {
    if (y <= l or r <= x) return {};
    if (x <= l and r <= y) return d;
    return ls->query(x, y) + rs->query(x, y);
  }
};
```

4.8 Blackmagic

```cpp
#include <bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
template<class T>
using BST = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
__gnu_pbds::priority_queue<node, decltype(cmp),
    pairing_heap_tag> pq(cmp);
gp_hash_table<int, gnu_pbds::priority_queue<node>::
    point_iterator> pqPos;
bst.insert((x << 20) + i);
bst.erase(bst.lower_bound(x << 20));
bst.order_of_key(x << 20) + 1;
*bst.find_by_order(x - 1) >> 20;
*--bst.lower_bound(x << 20) >> 20;
*bst.upper_bound((x + 1) << 20) >> 20;
```

4.9 Centroid Decomposition

```cpp
struct CenDec {
  vector<vector<pair<int, i64>>> G;
  vector<vector<i64>> pdis;
  vector<int> pa, ord, siz;
  vector<bool> vis;
  int getsiz(int u, int f) {
    siz[u] = 1;
    for (auto [v, w] : G[u]) if (v != f and !vis[v])
      siz[u] += getsiz(v, u);
    return siz[u];
  }
  int find(int u, int f, int s) {
    for (auto [v, w] : G[u]) if (v != f and !vis[v])
      if (siz[v] * 2 >= s) return find(v, u, s);
    return u;
  };
  void caldis(int u, int f, i64 dis) {
    pdis[u].push_back(dis);
    for (auto [v, w] : G[u]) if (v != f and !vis[v]) {
      caldis(v, u, dis + w);
    }
  }
  int build(int u = 0) {
    u = find(u, u, getsiz(u, u));
    ord.push_back(u);
    vis[u] = 1;
    for (auto [v, w] : G[u]) if (!vis[v]) {
      pa[build(v)] = u;
    }
    caldis(u, -1, 0); // if need
    vis[u] = 0;
    return u;
  };
  CenDec(int n) : G(n), pa(n, -1), vis(n), siz(n), pdis
    (n) {}
};
```

4.10 2D BIT

```cpp
template<class T>
struct BIT2D {
  vector<vector<T>> val;
  vector<vector<int>> Y;
  vector<int> X;
  int lowbit(int x) { return x & -x; }
  int getp(const vector<int> &v, int x) {
    return upper_bound(all(v), x) - v.begin();
  }
  BIT2D(vector<pair<int, int>> pos) {
    for (auto &[x, y] : pos) {
      X.push_back(x);
      swap(x, y);
    }
    sort(all(pos));
```

```
    sort(all(X));
    X.erase(unique(all(X)), X.end());
    Y.resize(X.size() + 1);
    val.resize(X.size() + 1);
    for (auto [y, x] : pos) {
      for (int i = getp(X, x); i <= X.size(); i +=
    lowbit(i))
        if (Y[i].empty() or Y[i].back() != y)
          Y[i].push_back(y);
    }
    for (int i = 1; i <= X.size(); i++) {
      val[i].assign(Y[i].size() + 1, T{});
    }
  }
  void add(int x, int y, T v) {
    for (int i = getp(X, x); i <= X.size(); i += lowbit
    (i))
      for (int j = getp(Y[i], y); j <= Y[i].size(); j
    += lowbit(j))
        val[i][j] += v;
  }
  T qry(int x, int y) {
    T r{};
    for (int i = getp(X, x); i > 0; i -= lowbit(i))
      for (int j = getp(Y[i], y); j > 0; j -= lowbit(j)
    ) {
        r += val[i][j];
      }
    return r;
  }
};
```

5 Dynamic Programming

5.1 CDQ

```
auto cmp2 = [&](int a, int b) -> bool { return P[a][1]
    < P[b][1]; };
auto cdq = [&](auto self, auto l, auto r) {
  if (r - l == 1) return;
  auto mid = l + (r - l) / 2;
  self(self, l, mid);
  auto tmp = vector<int>(mid, r);
  sort(l, mid, cmp2);
  sort(mid, r, cmp2);
  for (auto i = l, j = mid; j < r; j++) {
    while (i != mid and P[*i][1] < P[*j][1]) {
      bit.add(P[*i][2], dp[*i]);
      i++;
    }
    dp[*j].upd(bit.qry(P[*j][2]));
  };
  for (auto i = l; i < mid; i++) bit.reset(P[*i][2]);
  copy(all(tmp), mid);
  self(self, mid, r);
}; cdq(cdq, all(ord));
```

6 Math

6.1 Theorem

- Pick's theorem

  $A = i + \frac{b}{2} - 1$

- Laplacian matrix

  $L = D - A$

- Extended Catalan number

  $\frac{1}{(k-1)n+1}\binom{kn}{n}$

- Derangement $D_n = (n-1)(D_{n-1} + D_{n-2})$

- Möbius

  $\sum_{i|n} \mu(i) = [n = 1]$ $\sum_{i|n} \phi(i) = n$

- Inversion formula

  $f(n) = \sum_{i=0}^{n} \binom{n}{i} g(i)$ $g(n) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} f(i)$

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

- Sum of powers

  $\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k^+ \, n^{m+1-k}$

  $\sum_{j=0}^{m} \binom{m+1}{j} B_j^- = 0$

  note : $B_1^+ = -B_1^-$ $B_i^+ = B_i^-$

- Cipolla's algorithm

  $\left(\frac{u}{p}\right) = u^{\frac{p-1}{2}}$

  1. $\left(\frac{a^2 - n}{p}\right) = -1$
  2. $x = (a + \sqrt{a^2 - n})^{\frac{p+1}{2}}$

- Cayley's formula
  number of trees on n labeled vertices: n^{n-2}
  Let $T_{n,k}$ be the number of labelled forests on n vertices with k connected components, such that vertices 1, 2, ..., k all belong to different connected components. Then $T_{n,k} = kn^{n-k-1}$.

- High order residue

  $[d^{\frac{p-1}{(n,p-1)}} \equiv 1]$

- Packing and Covering

  |Maximum Independent Set| + |Minimum Vertex Cover| $= |V|$

- Kőnig's theorem

  |maximum matching| = |minimum vertex cover|

- Dilworth's theorem

  width = |largest antichain| = |smallest chain decomposition|

- Mirsky's theorem

  height = |longest chain| = |smallest antichain decomposition| = |minimum anticlique partition|

- Triangle center

  - $G : (1,)$
  - $O : (a^2(b^2 + c^2 - a^2),) = (sin2A,)$
  - $I : (a,) = (sinA)$
  - $E : (-a, b, c) = (-sinA, sinB, sinC)$
  - $H : (\frac{1}{b^2+c^2-a^2},) = (tanA,)$

- Lucas'Theorem :
  For $n, m \in \mathbb{Z}^*$ and prime P, $C(m, n) \mod P = \Pi(C(m_i, n_i))$ where m_i is the i-th digit of m in base P.

- Stirling approximation :
  $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n e^{\frac{1}{12n}}$

- Stirling Numbers(permutation $|P| = n$ with k cycles):
  $S(n, k) =$ coefficient of x^k in $\Pi_{i=0}^{n-1}(x + i)$

- Stirling Numbers(Partition n elements into k non-empty set):

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

- Pick's Theorem : $A = i + b/2 - 1$
  A: Area、i: grid number in the inner、b: grid number on the side

- Catalan number : $C_n = \binom{2n}{n}/(n+1)$
  $C_n^{n+m} - C_{n+1}^{n+m} = (m+n)!\frac{n-m+1}{n+1} \quad for \quad n \geq m$
  $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$
  $C_0 = 1 \quad and \quad C_{n+1} = 2(\frac{2n+1}{n+2})C_n$
  $C_0 = 1 \quad and \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \quad for \quad n \geq 0$

- Euler Characteristic:
  planar graph: $V - E + F - C = 1$
  convex polyhedron: $V - E + F = 2$
  V, E, F, C: number of vertices, edges, faces(regions), and components

- Kirchhoff's theorem :
  $A_{ii} = deg(i), A_{ij} = (i, j) \in E \, ? -1 : 0$, Deleting any one row, one column, and cal the det(A)

- Polya' theorem (c is number of color，m is the number of cycle size):
  $(\sum_{i=1}^{m} c^{gcd(i,m)})/m$

- Burnside lemma:
  $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$

- 錯排公式: (n 個人中，每個人皆不再原來位置的組合數):
  $dp[0] = 1; dp[1] = 0;$
  $dp[i] = (i-1) * (dp[i-1] + dp[i-2]);$

- Bell 數 (有 n 個人, 把他們拆組的方法總數) :
  $B_0 = 1$
  $B_n = \sum_{k=0}^{n} s(n,k) \quad (second - stirling)$
  $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$
- Wilson's theorem :
  $(p-1)! \equiv -1 (mod\, p)$
- Fermat's little theorem :
  $a^p \equiv a (mod\, p)$
- Euler's totient function:
  $A^{B^C} mod\, p = pow(A, pow(B, C, p-1)) mod\, p$
- 歐拉函數降冪公式:
  $A^B \text{ mod } C = A^{B \text{ mod } \phi(c) + \phi(c)} \text{ mod } C$
- 環相鄰塗異色:
  $(k-1)(-1)^n + (k-1)^n$
- 6 的倍數:
  $(a-1)^3 + (a+1)^3 + (-a)^3 + (-a)^3 = 6a$

6.2 Linear Sieve

```
template<size_t N>
struct Sieve {
  array<bool, N + 1> isp{};
  array<int, N + 1> mu{}, phi{};
  vector<int> primes{};
  Sieve() {
    isp.fill(true);
    isp[0] = isp[1] = false;
    mu[1] = 1;
    phi[1] = 1;
    for (int i = 2; i <= N; i++) {
      if (isp[i]) {
        primes.push_back(i);
        mu[i] = -1;
        phi[i] = i - 1;
      }
      for (i64 p : primes) {
        if (p * i > N) break;
        isp[p * i] = false;
        if (i % p == 0) {
          phi[p * i] = phi[i] * p;
          break;
        }
        phi[p * i] = phi[i] * (p - 1);
        mu[p * i] = mu[p] * mu[i];
      }
    }
  }
};
```

6.3 Exgcd

```
pair<i64, i64> exgcd(i64 a, i64 b) { // ax + by = 1
  if (b == 0) return {1, 0};
  auto [x, y] = exgcd(b, a % b);
  return {y, x - a / b * y};
};
```

6.4 CRT

```
i64 CRT(vector<pair<i64, i64>> E) {
  i128 R = 0, M = 1;
  for (auto [r, m] : E) {
    i128 d = r - R, g = gcd<i64>(M, m);
    if (d % g != 0) return -1;
    i128 x = exgcd(M / g, m / g).ff * d / g;
    R += M * x;
    M = M * m / g;
    R = (R % M + M) % M;
  }
  return R;
}
```

6.5 Factorize

```
struct Factorize {
  i64 fmul(i64 a, i64 b, i64 p) {
    return (i128)a * b % p;
  }
  i64 fpow(i64 a, i64 b, i64 p) {
    i64 res = 1;
    for (; b; b >>= 1, a = fmul(a, a, p))
      if (b & 1) res = fmul(res, a, p);
    return res;
  }
  bool Check(i64 a, i64 u, i64 n, int t) {
    a = fpow(a, u, n);
    if (a == 0 or a == 1 or a == n - 1) return true;
    for (int i = 0; i < t; i++) {
      a = fmul(a, a, n);
      if (a == 1) return false;
      if (a == n - 1) return true;
    }
    return false;
  };
  bool IsPrime(i64 n) {
    constexpr array<i64, 7> kChk{2, 235, 9375, 28178,
    450775, 9780504, 1795265022};
    // for int: {2, 7, 61}
    if (n < 2) return false;
    if (n % 2 == 0) return n == 2;
    i64 u = n - 1;
    int t = 0;
    while (u % 2 == 0) u >>= 1, t++;
    for (auto v : kChk) if (!Check(v, u, n, t)) return
    false;
    return true;
  }
  i64 PollardRho(i64 n) {
    if (n % 2 == 0) return 2;
    i64 x = 2, y = 2, d = 1, p = 1;
    auto f = [](i64 x, i64 n, i64 p) -> i64 {
      return ((i128)x * x % n + p) % n;
    };
    while (true) {
      x = f(x, n, p);
      y = f(f(y, n, p), n, p);
      d = __gcd(abs(x - y), n);
      if (d != n and d != 1) return d;
      if (d == n) ++p;
    }
  }
  i64 PrimeFactor(i64 n) {
    return IsPrime(n) ? n : PrimeFactor(PollardRho(n));
  }
};
```

6.6 NTT Prime List

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3

6.7 NTT

```
constexpr i64 cpow(i64 a, i64 b, i64 m) {
  i64 ret = 1;
  for (; b; b >>= 1, a = a * a % m)
    if (b & 1) ret = ret * a % m;
  return ret;
};
template<i64 M, i64 G>
struct NTT {
  static constexpr i64 iG = cpow(G, M - 2, M);
  void operator()(vector<i64> &v, bool inv) {
    int n = v.size();
    for (int i = 0, j = 0; i < n; i++) {
      if (i < j) swap(v[i], v[j]);
      for (int k = n / 2; (j ^= k) < k; k /= 2);
    }
    for (int mid = 1; mid < n; mid *= 2) {
      i64 w = cpow((inv ? iG : G), (M - 1) / (mid + mid
), M);
      for (int i = 0; i < n; i += mid * 2) {
        i64 now = 1;
        for (int j = i; j < i + mid; j++, now = now * w
 % M) {
          i64 x = v[j], y = v[j + mid];
          v[j] = (x + y * now) % M;
          v[j + mid] = (x - y * now) % M;
        }
      }
```

```
    }
    if (inv) {
      i64 in = cpow(n, M - 2, M);
      for (int i = 0; i < n; i++) v[i] = v[i] * in % M;
    }
  }
};
template<i64 M, i64 G>
vector<i64> convolution(vector<i64> f, vector<i64> g) {
  NTT<M, G> ntt;
  int sum = f.size() + g.size() - 1;
  int len = bit_ceil((u64)sum);
  f.resize(len); g.resize(len);
  ntt(f, 0), ntt(g, 0);
  for (int i = 0; i < len; i++) (f[i] *= g[i]) %= M;
  ntt(f, 1);
  f.resize(sum);
  for (int i = 0; i < sum; i++) if (f[i] < 0) f[i] += M
    ;
  return f;
}
vector<i64> convolution_ll(const vector<i64> &f, const
    vector<i64> &g) {
  constexpr i64 M1 = 998244353, G1 = 3;
  constexpr i64 M2 = 985661441, G2 = 3;
  constexpr i64 M1M2 = M1 * M2;
  constexpr i64 M1m1 = M2 * cpow(M2, M1 - 2, M1);
  constexpr i64 M2m2 = M1 * cpow(M1, M2 - 2, M2);
  auto c1 = convolution<M1, G1>(f, g);
  auto c2 = convolution<M2, G2>(f, g);
  for (int i = 0; i < c1.size(); i++) {
    c1[i] = ((i128)c1[i] * M1m1 + (i128)c2[i] * M2m2) %
     M1M2;
  }
  return c1;
}
```

6.8 FWT

1. XOR Convolution
   - $f(A) = (f(A_0) + f(A_1), f(A_0) - f(A_1))$
   - $f^{-1}(A) = (f^{-1}(\frac{A_0+A_1}{2}), f^{-1}(\frac{A_0-A_1}{2}))$
2. OR Convolution
   - $f(A) = (f(A_0), f(A_0) + f(A_1))$
   - $f^{-1}(A) = (f^{-1}(A_0), f^{-1}(A_1) - f^{-1}(A_0))$
3. AND Convolution
   - $f(A) = (f(A_0) + f(A_1), f(A_1))$
   - $f^{-1}(A) = (f^{-1}(A_0) - f^{-1}(A_1), f^{-1}(A_1))$

6.9 FWT

```
void ORop(i64 &x, i64 &y) { y = (y + x) % mod; }
void ORinv(i64 &x, i64 &y) { y = (y - x + mod) % mod; }

void ANDop(i64 &x, i64 &y) { x = (x + y) % mod; }
void ANDinv(i64 &x, i64 &y) { x = (x - y + mod) % mod;
    }

void XORop(i64 &x, i64 &y) { tie(x, y) = pair{(x + y) %
     mod, (x - y + mod) % mod}; }
void XORinv(i64 &x, i64 &y) { tie(x, y) = pair{(x + y)
    * inv2 % mod, (x - y + mod) * inv2 % mod}; }

void FWT(vector<i64> &f, auto &op) {
  const int s = f.size();
  for (int i = 1; i < s; i *= 2)
    for (int j = 0; j < s; j += i * 2)
      for (int k = 0; k < i; k++)
        op(f[j + k], f[i + j + k]);
}
// FWT(f, XORop), FWT(g, XORop)
// f[i] *= g[i]
// FWT(f, XORinv)
```

6.10 Lucas

```
// C(N, M) mod D
// 0 <= M <= N <= 10^18
// 1 <= D <= 10^6
i64 Lucas(i64 N, i64 M, i64 D) {
  auto Factor = [&](i64 x) -> vector<pair<i64, i64>> {
    vector<pair<i64, i64>> r;
    for (i64 i = 2; x > 1; i++)
      if (x % i == 0) {
        i64 c = 0;
        while (x % i == 0) x /= i, c++;
        r.emplace_back(i, c);
      }
    return r;
  };
  auto Pow = [&](i64 a, i64 b, i64 m) -> i64 {
    i64 r = 1;
    for (; b; b >>= 1, a = a * a % m)
      if (b & 1) r = r * a % m;
    return r;
  };
  vector<pair<i64, i64>> E;
  for (auto [p, q] : Factor(D)) {
    const i64 mod = Pow(p, q, 1 << 30);
    auto CountFact = [&](i64 x) -> i64 {
      i64 c = 0;
      while (x) c += (x /= p);
      return c;
    };
    auto CountBino = [&](i64 x, i64 y) { return
    CountFact(x) - CountFact(y) - CountFact(x - y); };
    auto Inv = [&](i64 x) -> i64 { return (exgcd(x, mod
    ).ff % mod + mod) % mod; };
    vector<i64> pre(mod + 1);
    pre[0] = pre[1] = 1;
    for (i64 i = 2; i <= mod; i++) pre[i] = (i % p == 0
     ? 1 : i) * pre[i - 1] % mod;
    function<i64(i64)> FactMod = [&](i64 n) -> i64 {
      if (n == 0) return 1;
      return FactMod(n / p) * Pow(pre[mod], n / mod,
    mod) % mod * pre[n % mod] % mod;
    };
    auto BinoMod = [&](i64 x, i64 y) -> i64 {
      return FactMod(x) * Inv(FactMod(y)) % mod * Inv(
    FactMod(x - y)) % mod;
    };
    i64 r = BinoMod(N, M) * Pow(p, CountBino(N, M), mod
    ) % mod;
    E.emplace_back(r, mod);
  };
  return CRT(E);
}
```

6.11 Berlekamp Massey

```
template <int P>
vector<int> BerlekampMassey(vector<int> x) {
 vector<int> cur, ls;
 int lf = 0, ld = 0;
 for (int i = 0; i < (int)x.size(); ++i) {
  int t = 0;
  for (int j = 0; j < (int)cur.size(); ++j)
   (t += 1LL * cur[j] * x[i - j - 1] % P) %= P;
  if (t == x[i]) continue;
  if (cur.empty()) {
   cur.resize(i + 1);
   lf = i, ld = (t + P - x[i]) % P;
   continue;
  }
  int k = 1LL * fpow(ld, P - 2, P) * (t + P - x[i]) % P
   ;
  vector<int> c(i - lf - 1);
  c.push_back(k);
  for (int j = 0; j < (int)ls.size(); ++j)
   c.push_back(1LL * k * (P - ls[j]) % P);
  if (c.size() < cur.size()) c.resize(cur.size());
  for (int j = 0; j < (int)cur.size(); ++j)
   c[j] = (c[j] + cur[j]) % P;
  if (i - lf + (int)ls.size() >= (int)cur.size()) {
   ls = cur, lf = i;
   ld = (t + P - x[i]) % P;
  }
  cur = c;
 }
 return cur;
}
```

6.12 Gauss Elimination

```
double Gauss(vector<vector<double>> &d) {
 int n = d.size(), m = d[0].size();
 double det = 1;
```

```
 for (int i = 0; i < m; ++i) {
  int p = -1;
  for (int j = i; j < n; ++j) {
   if (fabs(d[j][i]) < kEps) continue;
   if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p = j;
  }
  if (p == -1) continue;
  if (p != i) det *= -1;
  for (int j = 0; j < m; ++j) swap(d[p][j], d[i][j]);
  for (int j = 0; j < n; ++j) {
   if (i == j) continue;
   double z = d[j][i] / d[i][i];
   for (int k = 0; k < m; ++k) d[j][k] -= z * d[i][k];
  }
 }
 for (int i = 0; i < n; ++i) det *= d[i][i];
 return det;
}
```

6.13 Linear Equation

```
void linear_equation(vector<vector<double>> &d, vector<
    double> &aug, vector<double> &sol) {
  int n = d.size(), m = d[0].size();
  vector<int> r(n), c(m);
  iota(r.begin(), r.end(), 0);
  iota(c.begin(), c.end(), 0);
  for (int i = 0; i < m; ++i) {
    int p = -1, z = -1;
    for (int j = i; j < n; ++j) {
      for (int k = i; k < m; ++k) {
        if (fabs(d[r[j]][c[k]]) < eps) continue;
        if (p == -1 || fabs(d[r[j]][c[k]]) > fabs(d[r[p
]][c[z]])) p = j, z = k;
      }
    }
    if (p == -1) continue;
    swap(r[p], r[i]), swap(c[z], c[i]);
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      double z = d[r[j]][c[i]] / d[r[i]][c[i]];
      for (int k = 0; k < m; ++k) d[r[j]][c[k]] -= z *
    d[r[i]][c[k]];
      aug[r[j]] -= z * aug[r[i]];
    }
  }
  vector<vector<double>> fd(n, vector<double>(m));
  vector<double> faug(n), x(n);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) fd[i][j] = d[r[i]][c[j
]];
    faug[i] = aug[r[i]];
  }
  d = fd, aug = faug;
  for (int i = n - 1; i >= 0; --i) {
    double p = 0.0;
    for (int j = i + 1; j < n; ++j) p += d[i][j] * x[j
];
    x[i] = (aug[i] - p) / d[i][i];
  }
  for (int i = 0; i < n; ++i) sol[c[i]] = x[i];
}
```

6.14 LinearRec

```
template <int P>
int LinearRec(const vector<int> &s, const vector<int> &
    coeff, int k) {
  int n = s.size();
  auto Combine = [&](const auto &a, const auto &b) {
    vector<int> res(n * 2 + 1);
    for (int i = 0; i <= n; ++i) {
      for (int j = 0; j <= n; ++j)
        (res[i + j] += 1LL * a[i] * b[j] % P) %= P;
    }
    for (int i = 2 * n; i > n; --i) {
      for (int j = 0; j < n; ++j)
        (res[i - 1 - j] += 1LL * res[i] * coeff[j] % P)
     %= P;
    }
    res.resize(n + 1);
    return res;
  };
  vector<int> p(n + 1), e(n + 1);
  p[0] = e[1] = 1;
  for (; k > 0; k >>= 1) {
    if (k & 1) p = Combine(p, e);
    e = Combine(e, e);
  }
  int res = 0;
  for (int i = 0; i < n; ++i) (res += 1LL * p[i + 1] *
    s[i] % P) %= P;
  return res;
}
```

6.15 SubsetConv

```
vector<i64> SubsetConv(vector<i64> f, vector<i64> g) {
  const int n = f.size();
  const int U = __lg(n) + 1;
  vector F(U, vector<i64>(n));
  auto G = F, H = F;
  for (int i = 0; i < n; i++) {
    F[popcount<u64>(i)][i] = f[i];
    G[popcount<u64>(i)][i] = g[i];
  }
  for (int i = 0; i < U; i++) {
    FWT(F[i], ORop);
    FWT(G[i], ORop);
  }
  for (int i = 0; i < U; i++)
    for (int j = 0; j <= i; j++)
      for (int k = 0; k < n; k++)
        H[i][k] = (H[i][k] + F[i - j][k] * G[j][k]) %
    mod;
  for (int i = 0; i < U; i++) FWT(H[i], ORinv);
  for (int i = 0; i < n; i++) f[i] = H[popcount<u64>(i)
    ][i];
  return f;
}
```

6.16 SqrtMod

```
int SqrtMod(int n, int P) { // 0 <= x < P
  if (P == 2 or n == 0) return n;
  if (pow(n, (P - 1) / 2, P) != 1) return -1;
  mt19937 rng(12312);
  i64 z = 0, w;
  while (pow(w = (z * z - n + P) % P, (P - 1) / 2, P)
    != P - 1)
    z = rng() % P;
  const auto M = [P, w](auto &u, auto &v) {
    return make_pair(
      (u.ff * v.ff + u.ss * v.ss % P * w) % P,
      (u.ff * v.ss + u.ss * v.ff) % P
    );
  };
  pair<i64, i64> r(1, 0), e(z, 1);
  for (int w = (P + 1) / 2; w; w >>= 1, e = M(e, e))
    if (w & 1) r = M(r, e);
  return r.ff; // sqrt(n) mod P where P is prime
}
```

6.17 DiscreteLog

```
template<class T>
T BSGS(T x, T y, T M) {
 // x^? \equiv y (mod M)
 T t = 1, c = 0, g = 1;
 for (T M_ = M; M_ > 0; M_ >>= 1) g = g * x % M;
 for (g = gcd(g, M); t % g != 0; ++c) {
  if (t == y) return c;
  t = t * x % M;
 }
 if (y % g != 0) return -1;
 t /= g, y /= g, M /= g;
 T h = 0, gs = 1;
 for (; h * h < M; ++h) gs = gs * x % M;
 unordered_map<T, T> bs;
 for (T s = 0; s < h; bs[y] = ++s) y = y * x % M;
 for (T s = 0; s < M; s += h) {
  t = t * gs % M;
  if (bs.count(t)) return c + s + h - bs[t];
 }
 return -1;
}
```

6.18 FloorSum

```
// sigma 0 ~ n-1: (a * i + b) / m
i64 floor_sum(i64 n, i64 m, i64 a, i64 b) {
  u64 ans = 0;
  if (a < 0) {
    u64 a2 = (a % m + m) % m;
    ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
    a = a2;
  }
  if (b < 0) {
    u64 b2 = (b % m + m) % m;
    ans -= 1ULL * n * ((b2 - b) / m);
    b = b2;
  }
  while (true) {
    if (a >= m) {
      ans += n * (n - 1) / 2 * (a / m);
      a %= m;
    }
    if (b >= m) {
      ans += n * (b / m);
      b %= m;
    }
    u64 y_max = a * n + b;
    if (y_max < m) break;
    n = y_max / m;
    b = y_max % m;
    swap(m, a);
  }
  return ans;
}
```

6.19 Linear Programming Simplex

```
// max{cx} subject to {Ax<=b, x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// x = simplex(A, b, c); (A <= 100 x 100)
vector<double> simplex(
    const vector<vector<double>> &a,
    const vector<double> &b,
    const vector<double> &c) {

  int n = (int)a.size(), m = (int)a[0].size() + 1;
  vector val(n + 2, vector<double>(m + 1));
  vector<int> idx(n + m);
  iota(all(idx), 0);
  int r = n, s = m - 1;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j)
      val[i][j] = -a[i][j];
    val[i][m - 1] = 1;
    val[i][m] = b[i];
    if (val[r][m] > val[i][m])
      r = i;
  }
  copy(all(c), val[n].begin());
  val[n + 1][m - 1] = -1;
  for (double num; ; ) {
    if (r < n) {
      swap(idx[s], idx[r + m]);
      val[r][s] = 1 / val[r][s];
      for (int j = 0; j <= m; ++j) if (j != s)
        val[r][j] *= -val[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s)
          val[i][j] += val[r][j] * val[i][s];
        val[i][s] *= val[r][s];
      }
    }
    r = s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 || idx[s] > idx[j])
        if (val[n + 1][j] > eps || val[n + 1][j] > -eps
 && val[n][j] > eps)
          s = j;
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (val[i][s] < -eps) {
      if (r < 0
        || (num = val[r][m] / val[r][s] - val[i][m] /
 val[i][s]) < -eps
        || num < eps && idx[r + m] > idx[i + m])
        r = i;
    }
    if (r < 0) {
      //  Solution is unbounded.
      return vector<double>{};
    }
  }
  if (val[n + 1][m] < -eps) {
    //  No solution.
    return vector<double>{};
  }
  vector<double> x(m - 1);
  for (int i = m; i < n + m; ++i)
    if (idx[i] < m - 1)
      x[idx[i]] = val[i - m][m];
  return x;
}
```

6.20 Lagrange Interpolation

```
struct Lagrange {
  int deg{};
  vector<i64> C;
  Lagrange(const vector<i64> &P) {
    deg = P.size() - 1;
    C.assign(deg + 1, 0);

    for (int i = 0; i <= deg; i++) {
      i64 q = comb(-i) * comb(i - deg) % mod;
      if ((deg - i) % 2 == 1) {
        q = mod - q;
      }
      C[i] = P[i] * q % mod;
    }
  }
  i64 operator()(i64 x) { // 0 <= x < mod
    if (0 <= x and x <= deg) {
      i64 ans = comb(x) * comb(deg - x) % mod;
      if ((deg - x) % 2 == 1) {
        ans = (mod - ans);
      }
      return ans * C[x] % mod;
    }

    vector<i64> pre(deg + 1), suf(deg + 1);
    for (int i = 0; i <= deg; i++) {
      pre[i] = (x - i);
      if (i) {
        pre[i] = pre[i] * pre[i - 1] % mod;
      }
    }
    for (int i = deg; i >= 0; i--) {
      suf[i] = (x - i);
      if (i < deg) {
        suf[i] = suf[i] * suf[i + 1] % mod;
      }
    }

    i64 ans = 0;
    for (int i = 0; i <= deg; i++) {
      ans += (i == 0 ? 1 : pre[i - 1]) * (i == deg ? 1
: suf[i + 1]) % mod * C[i];
      ans %= mod;
    }

    if (ans < 0) ans += mod;

    return ans;
  }
};
```

7 Geometry

7.1 2D Point

```
using Pt = pair<double, double>;
using numbers::pi;
constexpr double eps = 1e-9;
Pt operator+(Pt a, Pt b) { return {a.ff + b.ff, a.ss +
    b.ss}; }
Pt operator-(Pt a, Pt b) { return {a.ff - b.ff, a.ss -
    b.ss}; }
```

```
Pt operator*(Pt a, double b) { return {a.ff * b, a.ss *
    b}; }
Pt operator/(Pt a, double b) { return {a.ff / b, a.ss /
    b}; }
double operator*(Pt a, Pt b) { return a.ff * b.ff + a.
    ss * b.ss; }
double operator^(Pt a, Pt b) { return a.ff * b.ss - a.
    ss * b.ff; }
double abs(Pt a) { return sqrt(a * a); }
double cro(Pt a, Pt b, Pt c) { return (b - a) ^ (c - a)
    ; }
int sig(double x) { return (x > -eps) - (x < eps); }
Pt rot(Pt u, double a) {
  Pt v{sin(a), cos(a)};
  return {u ^ v, u * v};
}
bool inedge(Pt a, Pt b, Pt c) {
  return ((a - b) ^ (c - b)) == 0 and (a - b) * (c - b)
    <= 0;
}
bool banana(Pt a, Pt b, Pt c, Pt d) {
  if (inedge(a, c, b) or inedge(a, d, b) or \
    inedge(c, a, d) or inedge(c, b, d))
    return true;
  return sig(cro(a, b, c)) * sig(cro(a, b, d)) < 0 and
    \
      sig(cro(c, d, a)) * sig(cro(c, d, b)) < 0;
}
Pt Inter(Pt a, Pt b, Pt c, Pt d) {
  double s = cro(c, d, a), t = -cro(c, d, b);
  return (a * t + b * s) / (s + t);
}
struct Line {
  Pt a{}, b{};
  Line() {}
  Line(Pt _a, Pt _b) : a{_a}, b{_b} {}
};
Pt Inter(Line L, Line R) {
  return Inter(L.a, L.b, R.a, R.b);
}
```

7.2 Convex Hull

```
vector<Pt> Hull(vector<Pt> P) {
  sort(all(P));
  P.erase(unique(all(P)), P.end());
  P.insert(P.end(), rall(P));
  vector<Pt> stk;
  for (auto p : P) {
    while (stk.size() >= 2 and \
        cro(*++stk.rbegin(), stk.back(), p) <= 0 and \
        (*++stk.rbegin() < stk.back()) == (stk.back() <
      p)) {
      stk.pop_back();
    }
    stk.push_back(p);
  }
  stk.pop_back();
  return stk;
}
```

7.3 Convex Hull trick

```
template<class T>
struct Convex {
  int n;
  vector<T> A, V, L, U;
  Convex(const vector<T> &_A) : A(_A), n(_A.size()) {
    // n >= 3
    auto it = max_element(all(A));
    L.assign(A.begin(), it + 1);
    U.assign(it, A.end()), U.push_back(A[0]);
    for (int i = 0; i < n; i++) {
      V.push_back(A[(i + 1) % n] - A[i]);
    }
  }
  int inside(T p, const vector<T> &h, auto f) { // 0:
    out, 1: on, 2: in
    auto it = lower_bound(all(h), p, f);
    if (it == h.end()) return 0;
    if (it == h.begin()) return p == *it;
    return 1 - sig(cro(*prev(it), p, *it));
  }
  int inside(T p) {
    return min(inside(p, L, less{}), inside(p, U,
    greater{}));
  }
  static bool cmp(T a, T b) { return sig(a ^ b) > 0; }
  int tangent(T v, bool close = true) {
    assert(v != T{});
    auto l = V.begin(), r = V.begin() + L.size() - 1;
    if (v < T{}) l = r, r = V.end();
    if (close) return (lower_bound(l, r, v, cmp) - V.
    begin()) % n;
    return (upper_bound(l, r, v, cmp) - V.begin()) % n;
  }
  array<int, 2> tangent2(T p) {
    array<int, 2> t{-1, -1};
    if (inside(p) == 2) return t;
    if (auto it = lower_bound(all(L), p); it != L.end()
     and p == *it) {
      int s = it - L.begin();
      return {(s + 1) % n, (s - 1 + n) % n};
    }
    if (auto it = lower_bound(all(U), p, greater{}); it
     != U.end() and p == *it) {
      int s = it - U.begin() + L.size() - 1;
      return {(s + 1) % n, (s - 1 + n) % n};
    }
    for (int i = 0; i != t[0]; i = tangent((A[t[0] = i]
     - p), 0));
    for (int i = 0; i != t[1]; i = tangent((p - A[t[1]
    = i]), 1));
    return t;
  }
  int Find(int l, int r, T a, T b) {
    if (r < l) r += n;
    int s = sig(cro(a, b, A[l % n]));
    while (r - l > 1) {
      (sig(cro(a, b, A[(l + r) / 2 % n])) == s ? l : r)
     = (l + r) / 2;
    }
    return l % n;
  };
  vector<int> LineIntersect(T a, T b) { // A_x A_x+1
    interset with ab
    assert(a != b);
    int l = tangent(a - b), r = tangent(b - a);
    if (sig(cro(a, b, A[l])) * sig(cro(a, b, A[r])) >=
    0) return {};
    return {Find(l, r, a, b), Find(r, l, a, b)};
  }
};
```

7.4 Dynamic Convex Hull

```
template<class T, class Comp = less<T>>
struct DynamicHull {
  set<T, Comp> H;
  DynamicHull() {}
  void insert(T p) {
    if (inside(p)) return;
    auto it = H.insert(p).ff;
    while (it != H.begin() and prev(it) != H.begin() \
        and cross(*prev(it, 2), *prev(it), *it) <= 0) {
      it = H.erase(--it);
    }
    while (it != --H.end() and next(it) != --H.end() \
        and cross(*it, *next(it), *next(it, 2)) <= 0) {
      it = --H.erase(++it);
    }
  }
  int inside(T p) { // 0: out, 1: on, 2: in
    auto it = H.lower_bound(p);
    if (it == H.end()) return 0;
    if (it == H.begin()) return p == *it;
    return 1 - sig(cross(*prev(it), p, *it));
  }
};
```

7.5 Half Plane Intersection

```
vector<Pt> HPI(vector<Line> P) {
  const int n = P.size();
  sort(all(P), [&](Line L, Line R) -> bool {
    Pt u = L.b - L.a, v = R.b - R.a;
    bool f = Pt(sig(u.ff), sig(u.ss)) < Pt{};
    bool g = Pt(sig(v.ff), sig(v.ss)) < Pt{};
```

```
    if (f != g) return f < g;
    return (sig(u ^ v) ? sig(u ^ v) : sig(cro(L.a, R.a,
      R.b))) > 0;
  });
  auto Same = [&](Line L, Line R) {
    Pt u = L.b - L.a, v = R.b - R.a;
    return sig(u ^ v) == 0 and sig(u * v) == 1;
  };
  deque <Pt> inter;
  deque <Line> seg;
  for (int i = 0; i < n; i++) if (i == 0 or !Same(P[i -
      1], P[i])) {
    while (seg.size() >= 2 and sig(cro(inter.back(), P[
      i].b, P[i].a)) == 1) {
      seg.pop_back(), inter.pop_back();
    }
    while (seg.size() >= 2 and sig(cro(inter[0], P[i].b
      , P[i].a)) == 1) {
      seg.pop_front(), inter.pop_front();
    }
    if (!seg.empty()) inter.push_back(Inter(seg.back(),
      P[i]));
    seg.push_back(P[i]);
  }
  while (seg.size() >= 2 and sig(cro(inter.back(), seg
    [0].b, seg[0].a)) == 1) {
    seg.pop_back(), inter.pop_back();
  }
  inter.push_back(Inter(seg[0], seg.back()));
  return vector<Pt>(all(inter));
}
```

7.6 Minimal Enclosing Circle

```
using circle = pair<Pt, double>;
struct MES {
  MES() {}
  bool inside(const circle &c, Pt p) {
    return abs(p - c.ff) <= c.ss + eps;
  };
  circle get_cir(Pt a, Pt b) {
    return circle((a + b) / 2., abs(a - b) / 2.);
  }
  circle get_cir(Pt a, Pt b, Pt c) {
    Pt p = (b - a) / 2.;
    p = Pt(-p.ss, p.ff);
    double t = ((c - a) * (c - b)) / (2 * (p * (c - a))
    );
    p = ((a + b) / 2.) + (p * t);
    return circle(p, abs(p - a));
  }
  circle get_mes(vector<Pt> P) {
    if (P.empty()) return circle{Pt(0, 0), 0};
    mt19937 rng(random_device{}());
    shuffle(all(P), rng);
    circle C{P[0], 0};
    for (int i = 1; i < P.size(); i++) {
      if (inside(C, P[i])) continue;
      C = get_cir(P[i], P[0]);
      for (int j = 1; j < i; j++) {
        if (inside(C, P[j])) continue;
        C = get_cir(P[i], P[j]);
        for (int k = 0; k < j; k++) {
          if (inside(C, P[k])) continue;
          C = get_cir(P[i], P[j], P[k]);
        }
      }
    }
    return C;
  }
};
```

7.7 Minkowski

```
vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) { // P
    , Q need sort
  const int n = P.size(), m = Q.size();
  P.push_back(P[0]), P.push_back(P[1]);
  Q.push_back(Q[0]), Q.push_back(Q[1]);
  vector<Pt> R;
  for (int i = 0, j = 0; i < n or j < m; ) {
    R.push_back(P[i] + Q[j]);
    auto v = (P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]);
    if (v >= 0) i++;
    if (v <= 0) j++;
  }
  return R;
}
```

7.8 TriangleCenter

```
Pt TriangleCircumCenter(Pt a, Pt b, Pt c) {
 Pt res;
 double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
 double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
 double ax = (a.x + b.x) / 2;
 double ay = (a.y + b.y) / 2;
 double bx = (c.x + b.x) / 2;
 double by = (c.y + b.y) / 2;
 double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by - ay)
    ) / (sin(a1) * cos(a2) - sin(a2) * cos(a1));
 return Pt(ax + r1 * cos(a1), ay + r1 * sin(a1));
}

Pt TriangleMassCenter(Pt a, Pt b, Pt c) {
 return (a + b + c) / 3.0;
}

Pt TriangleOrthoCenter(Pt a, Pt b, Pt c) {
 return TriangleMassCenter(a, b, c) * 3.0 -
    TriangleCircumCenter(a, b, c) * 2.0;
}

Pt TriangleInnerCenter(Pt a, Pt b, Pt c) {
 Pt res;
 double la = abs(b - c);
 double lb = abs(a - c);
 double lc = abs(a - b);
 res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb +
    lc);
 res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb +
    lc);
 return res;
}
```

7.9 Circle Triangle

```
double SectorArea(Pt a, Pt b, double r) {
  double theta = atan2(a.ss, a.ff) - atan2(b.ss, b.ff);
  while (theta <= 0) theta += 2 * pi;
  while (theta >= 2 * pi) theta -= 2 * pi;
  theta = min(theta, 2 * pi - theta);
  return r * r * theta / 2;
}

vector<Pt> CircleCrossLine(Pt a, Pt b, Pt o, double r)
    {
  double h = cro(o, a, b) / abs(a - b);
  Pt v = (a - b) / abs(a - b);
  Pt u = Pt{-v.ss, v.ff};
  Pt H = o + u * h;
  h = abs(h);
  vector<Pt> ret;
  if (sig(h - r) <= 0) {
    double d = sqrt(max(0., r * r - h * h));
    for (auto p : {H + (v * d), H - (v * d)})
      if (sig((a - p) * (b - p)) <= 0) {
        ret.push_back(p);
      }
  }
  return ret;
}

double AreaOfCircleTriangle(Pt a, Pt b, double r) {
  if (sig(abs(a) - r) <= 0 and sig(abs(b) - r) <= 0) {
    return abs(a ^ b) / 2;
  }
  if (abs(a) > abs(b)) swap(a, b);
  auto I = CircleCrossLine(a, b, {}, r);
  if (I.size() == 1) return abs(a ^ I[0]) / 2 +
    SectorArea(I[0], b, r);
  if (I.size() == 2) {
    return SectorArea(a, I[0], r) + SectorArea(I[1], b,
      r) + abs(I[0] ^ I[1]) / 2;
  }
  return SectorArea(a, b, r);
}
```

8 Stringology

8.1 KMP

```
vector<int> build_fail(string s) {
  const int len = s.size();
  vector<int> f(len, -1);
  for (int i = 1, p = -1; i < len; i++) {
    while (~p and s[p + 1] != s[i]) p = f[p];
    if (s[p + 1] == s[i]) p++;
    f[i] = p;
  }
  return f;
}
```

8.2 Z-algorithm

```
vector<int> zalgo(string s) {
  if (s.empty()) return {};
  int len = s.size();
  vector<int> z(len);
  z[0] = len;
  for (int i = 1, l = 1, r = 1; i < len; i++) {
    z[i] = i < r ? min(z[i - l], r - i) : 0;
    while (i + z[i] < len and s[i + z[i]] == s[z[i]]) z
    [i]++;
    if (i + z[i] > r) l = i, r = i + z[i];
  }
  return z;
}
```

8.3 Manacher

```
vector<int> manacher(const string &s) {
  string p = "@#";
  for (char c : s) p += c + '#';
  p += '$';
  vector<int> dp(p.size());
  int mid = 0, r = 1;
  for (int i = 1; i < p.size() - 1; i++) {
    auto &k = dp[i];
    k = i < mid + r ? min(dp[mid * 2 - i], mid + r - i)
     : 0;
    while (p[i + k + 1] == p[i - k - 1]) k++;
    if (i + k > mid + r) mid = i, r = k;
  }
  return vector<int>(dp.begin() + 2, dp.end() - 2);
}
```

8.4 SuffixArray Simple

```
struct SuffixArray {
  int n;
  vector<int> suf, rk, S;
  SuffixArray(vector<int> _S) : S(_S) {
    n = S.size();
    suf.assign(n, 0);
    rk.assign(n * 2, -1);
    iota(all(suf), 0);
    for (int i = 0; i < n; i++) rk[i] = S[i];
    for (int k = 2; k < n + n; k *= 2) {
      auto cmp = [&](int a, int b) -> bool {
        return rk[a] == rk[b] ? (rk[a + k / 2] < rk[b +
             k / 2]) : (rk[a] < rk[b]);
      };
      sort(all(suf), cmp);
      auto tmp = rk;
      tmp[suf[0]] = 0;
      for (int i = 1; i < n; i++) {
        tmp[suf[i]] = tmp[suf[i - 1]] + cmp(suf[i - 1],
      suf[i]);
      }
      rk.swap(tmp);
    }
  }
};
```

8.5 SuffixArray SAIS

```
namespace sfx {
#define fup(a, b) for (int i = a; i < b; i++)
#define fdn(a, b) for (int i = b - 1; i >= a; i--)
  constexpr int N = 5e5 + 5;
  bool _t[N * 2];
  int H[N], RA[N], x[N], _p[N];
  int SA[N * 2], _s[N * 2], _c[N * 2], _q[N * 2];
  void pre(int *sa, int *c, int n, int z) {
    fill_n(sa, n, 0), copy_n(c, z, x);
  }
  void induce(int *sa, int *c, int *s, bool *t, int n,
    int z) {
    copy_n(c, z - 1, x + 1);
    fup(0, n) if (sa[i] and !t[sa[i] - 1])
      sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
    copy_n(c, z, x);
    fdn(0, n) if (sa[i] and t[sa[i] - 1])
      sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
  }
  void sais(int *s, int *sa, int *p, int *q, bool *t,
    int *c, int n, int z) {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
    last = -1;
    fill_n(c, z, 0);
    fup(0, n) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) { fup(0, n) sa[--c[s[i]]] = i; return; }
    fdn(0, n - 1)
      t[i] = (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i
    + 1]);
    pre(sa, c, n, z);
    fup(1, n) if (t[i] and !t[i - 1])
      sa[--x[s[i]]] = p[q[i] = nn++] = i;
    induce(sa, c, s, t, n, z);
    fup(0, n) if (sa[i] and t[sa[i]] and !t[sa[i] - 1])
     {
      bool neq = last < 0 or !equal(s + sa[i], s + p[q[
    sa[i]] + 1], s + last);
      ns[q[last = sa[i]]] = nmxz += neq;
    }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz
     + 1);
    pre(sa, c, n, z);
    fdn(0, nn) sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
    induce(sa, c, s, t, n, z);
  }
  vector<int> build(vector<int> s, int n) {
    copy_n(begin(s), n, _s), _s[n] = 0;
    sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
    vector<int> sa(n);
    fup(0, n) sa[i] = SA[i + 1];
    return sa;
  }
  vector<int> lcp_array(vector<int> &s, vector<int> &sa
    ) {
    int n = int(s.size());
    vector<int> rnk(n);
    fup(0, n) rnk[sa[i]] = i;
    vector<int> lcp(n - 1);
    int h = 0;
    fup(0, n) {
      if (h > 0) h--;
      if (rnk[i] == 0) continue;
      int j = sa[rnk[i] - 1];
      for (; j + h < n and i + h < n; h++)
        if (s[j + h] != s[i + h]) break;
      lcp[rnk[i] - 1] = h;
    }
    return lcp;
  }
}
```

8.6 SuffixArray SAIS C++20

```
auto sais(const auto &s) {
  const int n = (int)s.size(), 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); t[n - 1] = 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] = int(lms.size());
    lms.push_back(sa[--x[s[i]]] = i);
  }
  induce(); vector<int> ns(lms.size());
  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;
  }
  ranges::fill(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;
}
// SPLIT_HASH_HERE 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, rev;
  Suffix(const auto &s) : n(int(s.size())),
    hi(n), rev(n) {
    vector<int> _s(n + 1); // _s[n] = 0
    copy(all(s), begin(_s)); // s shouldn't contain 0
    sa = sais(_s); sa.erase(sa.begin());
    for (int i = 0; i < n; i++) rev[sa[i]] = i;
    for (int i = 0, h = 0; i < n; i++) {
      if (!rev[i]) { h = 0; continue; }
      for (int j = sa[rev[i] - 1]; i + h < n && j + h <
       n
          && s[i + h] == s[j + h];) ++h;
      hi[rev[i]] = h ? h-- : 0;
    }
  }
};
```

8.7 Palindromic Tree

```
struct PAM {
  struct Node {
    int fail, len, dep;
    array<int, 26> ch;
    Node(int _len) : len{_len}, fail{}, ch{}, dep{} {};
  };
  vector<Node> g;
  vector<int> id;
  int odd, even, lst;
  string S;
  int new_node(int len) {
    g.emplace_back(len);
    return g.size() - 1;
  }
  PAM() : odd(new_node(-1)), even(new_node(0)) {
    lst = g[even].fail = odd;
  }
  int up(int p) {
    while (S.rbegin()[g[p].len + 1] != S.back())
      p = g[p].fail;
    return p;
  }
  int add(char c) {
    S += c;
    lst = up(lst);
    c -= 'a';
    if (!g[lst].ch[c]) g[lst].ch[c] = new_node(g[lst].
    len + 2);
    int p = g[lst].ch[c];
    g[p].fail = (lst == odd ? even : g[up(g[lst].fail)
    ].ch[c]);
    lst = p;
    g[lst].dep = g[g[lst].fail].dep + 1;
    id.push_back(lst);
    return lst;
  }
  void del() {
    S.pop_back();
    id.pop_back();
    lst = id.empty() ? odd : id.back();
  }
};
```

8.8 SmallestRotation

```
string Rotate(const string &s) {
 int n = s.length();
 string t = s + s;
 int i = 0, j = 1;
 while (i < n && j < n) {
  int k = 0;
  while (k < n && t[i + k] == t[j + k]) ++k;
  if (t[i + k] <= t[j + k]) j += k + 1;
  else i += k + 1;
  if (i == j) ++j;
 }
 int pos = (i < n ? i : j);
 return t.substr(pos, n);
}
```

8.9 Aho-Corasick

```
struct ACauto {
  static const int sigma = 26;
  struct Node {
    array<Node*, sigma> ch{};
    Node *fail = nullptr;
    int cnt = 0;
    vector<int> id;
  } *root;
  ACauto() : root(new Node()) {}
  void insert(const string &s, int id) {
    auto p = root;
    for (char c : s) {
      int d = c - 'a';
      if (!p->ch[d]) p->ch[d] = new Node();
      p = p->ch[d];
    }
    p->id.emplace_back(id);
  }
  vector<Node*> ord;
  void build() {
    root->fail = root;
    queue<Node*> que;
    for (int i = 0; i < sigma; i++) {
      if (root->ch[i]) {
        root->ch[i]->fail = root;
        que.emplace(root->ch[i]);
      }
      else {
        root->ch[i] = root;
      }
    }
    while (!que.empty()) {
      auto p = que.front(); que.pop();
      ord.emplace_back(p);
      for (int i = 0; i < sigma; i++) {
        if (p->ch[i]) {
          p->ch[i]->fail = p->fail->ch[i];
          que.emplace(p->ch[i]);
        }
        else {
          p->ch[i] = p->fail->ch[i];
        }
      }
    }
  }
  void walk(const string &s) {
    auto p = root;
    for (const char &c : s) {
      int d = c - 'a';
      (p = p->ch[d])->cnt++;
    }
  }
  void count(vector<int> &cnt) {
    reverse(all(ord));
    for (auto p : ord) {
      p->fail->cnt += p->cnt;
      for (int id : p->id)
        cnt[id] = p->cnt;
    }
```

```
  }
};
```

8.10 Suffix Automaton

```
struct SAM {
  struct Node {
    int link{}, len{};
    array<int, 26> ch{};
  };
  vector<Node> n;
  int lst = 0;
  SAM() : n(1) {}
  int newNode() {
    n.emplace_back();
    return n.size() - 1;
  }
  void reset() {
    lst = 0;
  }
  int add(int c) {
    if (n[n[lst].ch[c]].len == n[lst].len + 1) { //
    General
      return lst = n[lst].ch[c];
    }
    int cur = newNode();
    n[cur].len = n[lst].len + 1;
    while (lst != 0 and n[lst].ch[c] == 0) {
      n[lst].ch[c] = cur;
      lst = n[lst].link;
    }
    int p = n[lst].ch[c];
    if (p == 0) {
      n[cur].link = 0;
      n[0].ch[c] = cur;
    } else if (n[p].len == n[lst].len + 1) {
      n[cur].link = p;
    } else {
      int t = newNode();
      n[t] = n[p];
      n[t].len = n[lst].len + 1;
      while (n[lst].ch[c] == p) {
        n[lst].ch[c] = t;
        lst = n[lst].link;
      }
      n[p].link = n[cur].link = t;
    }
    return lst = cur;
  }
};
```

9 Misc

9.1 Fraction Binary Search

```
// Binary search on Stern-Brocot Tree
// Parameters: n, pred
// n: Q_n is the set of all rational numbers whose
    denominator does not exceed n
// pred: pair<i64, i64> -> bool, pred({0, 1}) must be
    true
// Return value: {{a, b}, {x, y}}
// a/b is bigger value in Q_n that satisfy pred()
// x/y is smaller value in Q_n that not satisfy pred()
// Complexity: O(log^2 n)
using Pt = pair<i64, i64>;
Pt operator+(Pt a, Pt b) { return {a.ff + b.ff, a.ss +
    b.ss}; }
Pt operator*(i64 a, Pt b) { return {a * b.ff, a * b.ss
    }; }
pair<pair<i64, i64>, pair<i64, i64>> FractionSearch(i64
     n, const auto &pred) {
  pair<i64, i64> low{0, 1}, hei{1, 0};
  while (low.ss + hei.ss <= n) {
    bool cur = pred(low + hei);
    auto &fr{cur ? low : hei}, &to{cur ? hei : low};
    u64 L = 1, R = 2;
    while ((fr + R * to).ss <= n and pred(fr + R * to)
    == cur) {
      L *= 2;
      R *= 2;
    }
    while (L + 1 < R) {
      u64 M = (L + R) / 2;
      ((fr + M * to).ss <= n and pred(fr + M * to) ==
    cur ? L : R) = M;
    }
    fr = fr + L * to;
  }
  return {low, hei};
}
```

9.2 de Bruijn sequence

```
constexpr int MAXC = 10, MAXN = 1e5 + 10;
struct DBSeq {
  int C, N, K, L;
  int buf[MAXC * MAXN];
  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) { //
    alphabet, len, k
    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;
```

9.3 HilbertCurve

```
long long hilbert(int n, int x, int y) {
 long long 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;
}
```

9.4 DLX

```
namespace dlx {
int lt[maxn], rg[maxn], up[maxn], dn[maxn], cl[maxn],
    rw[maxn], bt[maxn], s[maxn], head, sz, ans;
void init(int 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(int r, 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] = r, cl[v] = c;
  ++s[c];
  if (i > 0) lt[v] = v - 1;
 }
 lt[f] = sz - 1;
}
void remove(int c) {
 lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
```

```
 for (int i = dn[c]; i != c; i = dn[i]) {
  for (int j = rg[i]; j != i; j = rg[j])
   up[dn[j]] = up[j], dn[up[j]] = dn[j], --s[cl[j]];
 }
}
void restore(int c) {
 for (int i = up[c]; i != c; i = up[i]) {
  for (int j = lt[i]; j != i; j = lt[j])
   ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
 }
 lt[rg[c]] = c, rg[lt[c]] = c;
}
// Call dlx::make after inserting all rows.
void make(int c) {
 for (int i = 0; i < c; ++i)
  dn[bt[i]] = i, up[i] = bt[i];
}
void dfs(int dep) {
 if (dep >= ans) return;
 if (rg[head] == head) return ans = dep, void();
 if (dn[rg[head]] == rg[head]) return;
 int c = rg[head];
 int w = c;
 for (int x = c; x != head; x = rg[x]) if (s[x] < s[w])
     w = x;
 remove(w);
 for (int i = dn[w]; i != w; i = dn[i]) {
  for (int j = rg[i]; j != i; j = rg[j]) remove(cl[j]);
  dfs(dep + 1);
  for (int j = lt[i]; j != i; j = lt[j]) restore(cl[j])
     ;
 }
 restore(w);
}
int solve() {
 ans = 1e9, dfs(0);
 return ans;
}}
```

9.5 NextPerm

```
i64 next_perm(i64 x) {
  i64 y = x | (x - 1);
  return (y + 1) | (((~y & -~y) - 1) >> (__builtin_ctz(
    x) + 1));
}
```

9.6 FastIO

```
struct FastIO {
  const static int ibufsiz = 4<<20, obufsiz = 18<<20;
  char ibuf[ibufsiz], *ipos = ibuf, obuf[obufsiz], *
    opos = obuf;
  FastIO() { fread(ibuf, 1, ibufsiz, stdin); }
  ~FastIO() { fwrite(obuf, 1, opos - obuf, stdout); }
  template<class T> FastIO& operator>>(T &x) {
    bool sign = 0; while (!isdigit(*ipos)) { if (*ipos
    == '-') sign = 1; ++ipos; }
    x = *ipos++ & 15;
    while (isdigit(*ipos)) x = x * 10 + (*ipos++ & 15);
    if (sign) x = -x;
    return *this;
  }
  template<class T> FastIO& operator<<(T n) {
    static char _buf[18];
    char* _pos = _buf;
    if (n < 0) *opos++ = '-', n = -n;
    do *_pos++ = '0' + n % 10; while (n /= 10);
    while (_pos != _buf) *opos++ = *--_pos;
    return *this;
  }
  FastIO& operator<<(char ch) { *opos++ = ch; return *
    this; }
} FIO;
#define cin FIO
#define cout FIO
```

9.7 Python FastIO

```
import sys
sys.stdin.readline()
sys.stdout.write()
```

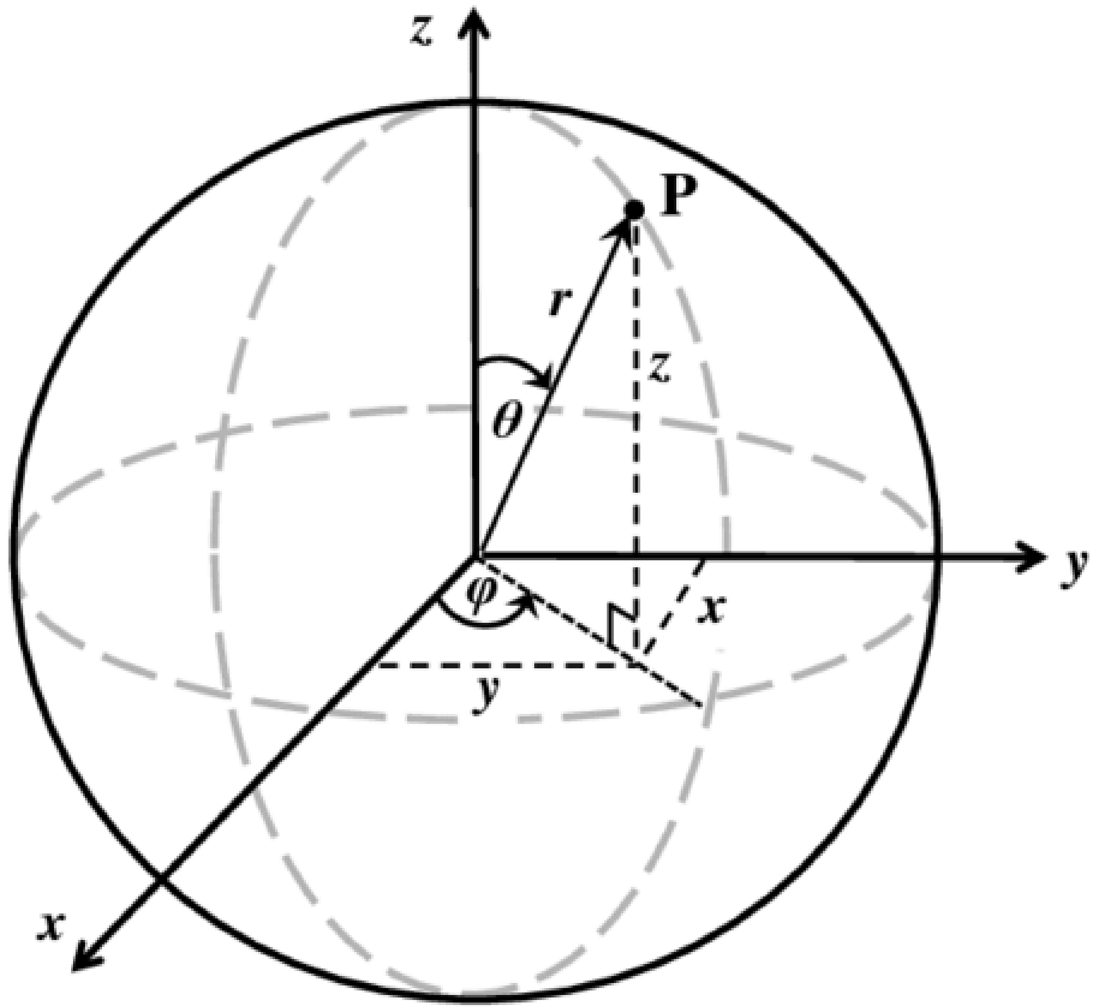
9.8 Trick

```
dp[61][0][0][0][7] = 1;
for (int h = 60; h >= 0; h--) {
  int s = (n >> h & 1) * 7;
  for (int x = 0; x < 8; x++) if (__builtin_parity(x)
    == 0) {
    for (int y = 0; y < 8; y++)
      if (((y & ~s) & x) == 0) {
        for (int a = 0; a < A[0]; a++)
          for (int b = 0; b < A[1]; b++)
            for (int c = 0; c < A[2]; c++) {
              if (dp[h + 1][a][b][c][y] == 0) continue;
              i64 i = ((x >> 2 & 1LL) << h) % A[0];
              i64 j = ((x >> 1 & 1LL) << h) % A[1];
              i64 k = ((x >> 0 & 1LL) << h) % A[2];
              auto &val =
              dp[h][(i + a) % A[0]][(j + b) % A[1]][(k
    + c) % A[2]][y & ~(s ^ x)];
              val = add(val, dp[h + 1][a][b][c][y]);
            }
      }
  }
}
pair<i64, i64> Split(i64 x) {
  if (x == 1) return {0, 0};
  i64 h = __lg(x);
  i64 fill = (1LL << (h + 1)) - 1;
  i64 l = (1LL << h) - 1 - max(0LL, fill - x - (1LL <<
    (h - 1)));
  i64 r = x - 1 - l;
  return {l, r};
};
{
  auto [ls, l] = DP(lo);
  auto [rs, r] = DP(hi);
  if (r < K) {
    cout << "Impossible\n";
    return;
  }
  if (l == K) cout << ls << '\n';
  else if (r == K) cout << rs << '\n';
  else {
    cout << (ls * (r - K) + rs * (K - l)) / (r - l) <<
    '\n';
  }
}
{
  auto F = [&](int L, int R) -> i64 {
    static vector<int> cnt(n);
    static int l = 0, r = -1;
    static i64 ans = 0;

    auto Add = [&](int x) {
      ans += cnt[A[x]]++;
    };
    auto Del = [&](int x) {
      ans -= --cnt[A[x]];
    };

    while (r < R) Add(++r);
    while (L < l) Add(--l);
    while (R < r) Del(r--);
    while (l < L) Del(l++);

    return ans;
  };

  vector<i64> dp(n), tmp(n);
  function<void(int, int, int, int)> sol = [&](int l,
    int r, int x, int y) {
    if (l > r) return;
    int mid = (l + r) / 2;
    int z = mid;
    for (int i = min(y, mid - 1); i >= x; i--)
      if (chmin(tmp[mid], dp[i] + F(i + 1, mid))) {
        z = i;
      }
    if (l == r) return;
    sol(l, mid - 1, x, z);
    sol(mid + 1, r, z, y);
  };
```

```
  for (int i = 0; i < n; i++)
    dp[i] = F(0, i);

  for (int i = 2; i <= m; i++) {
    tmp.assign(n, inf<i64>);
    sol(0, n - 1, 0, n - 1);
    dp = tmp;
  }

  cout << dp[n - 1] << '\n';
}
```

9.9 PyTrick

```
from itertools import permutations
op = ['+', '-', '*', '']
a, b, c, d = input().split()
ans = set()
for (x,y,z,w) in permutations([a, b, c, d]):
  for op1 in op:
    for op2 in op:
      for op3 in op:
        val = eval(f"{x}{op1}{y}{op2}{z}{op3}{w}")
        if (op1 == '' and op2 == '' and op3 == '') or
            val < 0:
          continue
        ans.add(val)
print(len(ans))
#
from decimal import *
from fractions import *
s = input()
n = int(input())
f = Fraction(s)
g = Fraction(s).limit_denominator(n)
h = f * 2 - g
if h.numerator <= n and h.denominator <= n and h < g:
  g = h
print(g.numerator, g.denominator)

from fractions import Fraction
x = Fraction(1, 2), y = Fraction(1)
print(x.as_integer_ratio()) # print 1/2
print(x.is_integer())
print(x.__round__())
print(float(x))

r = Fraction(input())
N = int(input())
r2 = r - 1 / Fraction(N) ** 2
ans = r.limit_denominator(N)
ans2 = r2.limit_denominator(N)
if ans2 < ans and 0 <= ans2 <= 1 and abs(ans - r) >=
    abs(ans2 - r):
  ans = ans2
print(ans.numerator,ans.denominator)
```



$$x = r \sin\theta \cos\varphi$$

$$y = r \sin\theta \sin\varphi$$

$$z = r \cos\theta$$

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \cos^{-1}(z/r)$$

$$\varphi = \tan^{-1}(y/x)$$