Contents

Contents

1 Basic

1.1 vimrc

```
se is nu ru et tgc sc hls cin cino+=j1 sw=4 sts=4 bs=2 mouse=a "encoding=utf-8 ls=2
syn on | colo desert | filetype indent on
map <leader>b <ESC>:w<CR>:!g++ "%" -o "%<" -g -std=gnu++20 -DCKISEKI -Wall -Wextra -Wshadow -Wfatal-errors -Wconversion -fsanitize=address,undefined,float-divide-by-zero,float-cast-overflow && echo success<CR>
map <leader>z <ESC>:w<CR>:!g++ "%" -o "%<" -O2 -g -std=gnu++20 && echo success<CR>
map <leader>i <ESC>:!./"%<"<CR>
map <leader>r <ESC>:!cat 01.in && echo "---" && ./"%<" < 01.in<CR>
map <leader>l :%d<bar>0r ~/t.cpp<CR>
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d "[:space:]" \| md5sum \| cut -c-6
let c_no_curly_error=1
```

1.2 Default code

```
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
using ll = long long;
#define SZ(v) (ll)((v).size())
#define pb emplace_back
#define AI(i) begin(i), end(i)
#define X first
#define Y second
template<class T> bool chmin(T &a, T b) { return b < a && (a = b, true); }
template<class T> bool chmax(T &a, T b) { return a < b && (a = b, true); }
#ifdef KEV
#define DE(args...) kout("[ " + string(#args) + " ] = ", args)
void kout() { cerr << endl; }
template<class T, class ...U> void kout(T a, U ...b) { cerr << a << ' ', kout(b...); }
template<class T> void debug(T l, T r) { while (l != r) cerr << *l << " \n"[next(l)==r], ++l; }
#else
#define DE(...) 0
#define debug(...) 0
#endif
int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  return 0;
}
```

1.3 Fast Integer Input

```
char buf[1 << 16], *p1 = buf, *p2 = buf;
char get() {
  if (p1 == p2) {
    p1 = buf;
    p2 = p1 + fread(buf, 1, sizeof(buf), stdin);
  }
  if (p1 == p2)
    return -1;
  return *p1++;
}
char readChar() {
  char c = get();
  while (isspace(c))
    c = get();
  return c;
}
int readInt() {
  int x = 0;
  char c = get();
  while (!isdigit(c))
    c = get();
  while (isdigit(c)) {
    x = 10 * x + c - '0';
    c = get();
  }
  return x;
}
```

1.4 Fast Python Input

```
import sys, os, io
input = io.BytesIO(os.read(0, os.fstat(0).st_size)).readline
```

1.5 Pragma optimization

```
#pragma GCC optimize("Ofast", "no-stack-protector", "no-math-errno", "unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.2,popcnt,abm,mmx,avx,tune=native,arch=core-avx2,tune=core-avx2")
#pragma GCC ivdep
```

2 Flows, Matching

2.1 Flow

```
template <typename F>
struct Flow {
  static constexpr F INF = numeric_limits<F>::max() / 2;
  struct Edge {
    int to;
    F cap;
    Edge(int to, F cap) : to(to), cap(cap) {}
  };
  int n;
  vector<Edge> e;
  vector<vector<int>> adj;
  vector<int> cur, h;
  Flow(int n) : n(n), adj(n) {}
  bool bfs(int s, int t) {
    h.assign(n, -1);
    queue<int> q;
    h[s] = 0;
    q.push(s);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int i : adj[u]) {
        auto [v, c] = e[i];
        if (c > 0 && h[v] == -1) {
          h[v] = h[u] + 1;
          if (v == t) { return true; }
          q.push(v);
        }
      }
    }
    return false;
  }
  F dfs(int u, int t, F f) {
    if (u == t) { return f; }
    F r = f;
    for (int &i = cur[u]; i < int(adj[u].size()); i++) {
      int j = adj[u][i];
      auto [v, c] = e[j];
      if (c > 0 && h[v] == h[u] + 1) {
        F a = dfs(v, t, min(r, c));
        e[j].cap -= a;
        e[j ^ 1].cap += a;
        r -= a;
        if (r == 0) { return f; }
      }
    }
    return f - r;
  }
  // can be bidirectional
  void addEdge(int u, int v, F cf = INF, F cb = 0) {
    adj[u].push_back(e.size()), e.emplace_back(v, cf);
    adj[v].push_back(e.size()), e.emplace_back(u, cb);
  }
  F maxFlow(int s, int t) {
    F ans = 0;
    while (bfs(s, t)) {
      cur.assign(n, 0);
      ans += dfs(s, t, INF);
    }
    return ans;
  }
  // do max flow first
  vector<int> minCut() {
    vector<int> res(n);
    for (int i = 0; i < n; i++) { res[i] = h[i] != -1; }
    return res;
  }
};
```

2.2 MCMF

```
template <class Flow, class Cost>
struct MinCostMaxFlow {
public:
  static constexpr Flow flowINF = numeric_limits<Flow>::max();
  static constexpr Cost costINF = numeric_limits<Cost>::max();
  MinCostMaxFlow() {}
  MinCostMaxFlow(int n) : n(n), g(n) {}
  int addEdge(int u, int v, Flow cap, Cost cost) {
    int m = int(pos.size());
    pos.push_back({u, int(g[u].size())});
    g[u].push_back({v, int(g[v].size()), cap, cost});
    g[v].push_back({u, int(g[u].size()) - 1, 0, -cost});
    return m;
  }
  struct edge {
    int u, v;
    Flow cap, flow;
    Cost cost;
  };
  edge getEdge(int i) {
    auto _e = g[pos[i].first][pos[i].second];
    auto _re = g[_e.v][_e.rev];
    return {pos[i].first, _e.v, _e.cap + _re.cap, _re.cap, _e.cost};
  }
  vector<edge> edges() {
    int m = int(pos.size());
    vector<edge> result(m);
    for (int i = 0; i < m; i++) { result[i] = getEdge(i); }
    return result;
  }
  pair<Flow, Cost> maxFlow(int s, int t, Flow flow_limit = flowINF) { return slope(s, t, flow_limit).back(); }
  vector<pair<Flow, Cost>> slope(int s, int t, Flow flow_limit = flowINF) {
    vector<Cost> dual(n, 0), dis(n);
    vector<int> pv(n), pe(n), vis(n);
    auto dualRef = [&]() {
      fill(dis.begin(), dis.end(), costINF);
      fill(pv.begin(), pv.end(), -1);
      fill(pe.begin(), pe.end(), -1);
      fill(vis.begin(), vis.end(), false);
      struct Q {
        Cost key;
        int u;
        bool operator<(Q o) const { return key > o.key; }
      };
```

```
      priority_queue<Q> h;
      dis[s] = 0;
      h.push({0, s});
      while (!h.empty()) {
        int u = h.top().u;
        h.pop();
        if (vis[u]) { continue; }
        vis[u] = true;
        if (u == t) { break; }
        for (int i = 0; i < int(g[u].size()); i++) {
          auto e = g[u][i];
          if (vis[e.v] || e.cap == 0) continue;
          Cost cost = e.cost - dual[e.v] + dual[u];
          if (dis[e.v] - dis[u] > cost) {
            dis[e.v] = dis[u] + cost;
            pv[e.v] = u;
            pe[e.v] = i;
            h.push({dis[e.v], e.v});
          }
        }
      }
      if (!vis[t]) { return false; }
      for (int v = 0; v < n; v++) {
        if (!vis[v]) continue;
        dual[v] -= dis[t] - dis[v];
      }
      return true;
    };
    Flow flow = 0;
    Cost cost = 0, prevCost = -1;
    vector<pair<Flow, Cost>> result;
    result.push_back({flow, cost});
    while (flow < flow_limit) {
      if (!dualRef()) break;
      Flow c = flow_limit - flow;
      for (int v = t; v != s; v = pv[v]) {
        c = min(c, g[pv[v]][pe[v]].cap);
      }
      for (int v = t; v != s; v = pv[v]) {
        auto& e = g[pv[v]][pe[v]];
        e.cap -= c;
        g[v][e.rev].cap += c;
      }
      Cost d = -dual[s];
      flow += c;
      cost += c * d;
      if (prevCost == d) { result.pop_back(); }
      result.push_back({flow, cost});
      prevCost = cost;
    }
    return result;
  }
private:
  int n;
  struct _edge {
    int v, rev;
    Flow cap;
    Cost cost;
  };
  vector<pair<int, int>> pos;
  vector<vector<_edge>> g;
};
```

2.3 GomoryHu Tree

```
auto gomory(int n, vector<array<int, 3>> e) {
  Flow<int, int> mf(n);
  for (auto [u, v, c] : e) { mf.addEdge(u, v, c, c); }
  vector<array<int, 3>> res;
  vector<int> p(n);
  for (int i = 1; i < n; i++) {
    for (int j = 0; j < int(e.size()); j++) { mf.e[j << 1].cap = mf.e[j << 1 | 1].cap = e[j][2]; }
    int f = mf.maxFlow(i, p[i]);
    auto cut = mf.minCut();
    for (int j = i + 1; j < n; j++) { if (cut[i] == cut[j] && p[i] == p[j]) { p[j] = i; }}
    res.push_back({f, i, p[i]});
  }
  return res;
}
```

2.4 Global Minimum Cut

```
// O(V ^ 3)
template <typename F>
struct GlobalMinCut {
  static constexpr int INF = numeric_limits<F>::max() / 2;
  int n;
  vector<int> vis, wei;
  vector<vector<int>> adj;
  GlobalMinCut(int n) : n(n), vis(n), wei(n), adj(n, vector<int>(n)) {}
  void addEdge(int u, int v, int w){
    adj[u][v] += w;
    adj[v][u] += w;
  }
  int solve() {
    int sz = n;
    int res = INF, x = -1, y = -1;
    auto search = [&]() {
      fill(vis.begin(), vis.begin() + sz, 0);
      fill(wei.begin(), wei.begin() + sz, 0);
      x = y = -1;
      int mx, cur;
      for (int i = 0; i < sz; i++) {
        mx = -1, cur = 0;
        for (int j = 0; j < sz; j++) {
          if (wei[j] > mx) {
            mx = wei[j], cur = j;
          }
        }
        vis[cur] = 1, wei[cur] = -1;
        x = y;
        y = cur;
        for (int j = 0; j < sz; j++) {
          if (!vis[j]) {
            wei[j] += adj[cur][j];
          }
        }
      }
      return mx;
    };
    while (sz > 1) {
      res = min(res, search());
      for (int i = 0; i < sz; i++) {
        adj[x][i] += adj[y][i];
        adj[i][x] = adj[x][i];
      }
      for (int i = 0; i < sz; i++) {
        adj[y][i] = adj[sz - 1][i];
        adj[i][y] = adj[i][sz - 1];
      }
      sz--;
    }
    return res;
  }
};
```

2.5 Bipartite Matching

```
struct BipartiteMatching {
  int n, m;
  vector<vector<int>> adj;
  vector<int> l, r, dis, cur;
  BipartiteMatching(int n, int m) : n(n), m(m), adj(n), l(n, -1), r(m, -1), dis(n), cur(n) {}
  void addEdge(int u, int v) { adj[u].push_back(v); }
  void bfs() {
    vector<int> q;
    for (int u = 0; u < n; u++) {
      if (l[u] == -1) {
        q.push_back(u), dis[u] = 0;
      } else {
        dis[u] = -1;
      }
    }
    for (int i = 0; i < int(q.size()); i++) {
      int u = q[i];
      for (auto v : adj[u]) {
        if (r[v] != -1 && dis[r[v]] == -1) {
          dis[r[v]] = dis[u] + 1;
          q.push_back(r[v]);
        }
      }
    }
  }
  bool dfs(int u) {
    for (int &i = cur[u]; i < int(adj[u].size()); i++) {
      int v = adj[u][i];
      if (r[v] == -1 || dis[r[v]] == dis[u] + 1 && dfs(r[v])) {
        l[u] = v, r[v] = u;
        return true;
      }
    }
    return false;
  }
  int maxMatching() {
    int match = 0;
    while (true) {
      bfs();
      fill(cur.begin(), cur.end(), 0);
      int cnt = 0;
      for (int u = 0; u < n; u++) {
        if (l[u] == -1) {
          cnt += dfs(u);
        }
      }
      if (cnt == 0) {
        break;
      }
      match += cnt;
    }
    return match;
  }
  auto minVertexCover() {
    vector<int> L, R;
    for (int u = 0; u < n; u++) {
      if (dis[u] == -1) {
        L.push_back(u);
      } else if (l[u] != -1) {
        R.push_back(l[u]);
      }
    }
    return pair(L, R);
  }
};
```

2.6 GeneralMatching

```
struct GeneralMatching {
  int n;
  vector<vector<int>> adj;
  vector<int> match;
  GeneralMatching(int n) : n(n), adj(n), match(n, -1) {}
  void addEdge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  int maxMatching() {
    vector<int> vis(n), link(n), f(n), dep(n);
    auto find = [&](int u) {
      while (f[u] != u) { u = f[u] = f[f[u]]; }
      return u;
    };
    auto lca = [&](int u, int v) {
      u = find(u);
      v = find(v);
      while (u != v) {
        if (dep[u] < dep[v]) { swap(u, v); }
        u = find(link[match[u]]);
      }
      return u;
    };
    queue<int> q;
    auto blossom = [&](int u, int v, int p) {
      while (find(u) != p) {
```

```
        link[u] = v;
        v = match[u];
        if (vis[v] == 0) {
          vis[v] = 1;
          q.push(v);
        }
        f[u] = f[v] = p;
        u = link[v];
      }
    };
    auto augment = [&](int u) {
      while (!q.empty()) { q.pop(); }
      iota(f.begin(), f.end(), 0);
      fill(vis.begin(), vis.end(), -1);
      q.push(u), vis[u] = 1, dep[u] = 0;
      while (!q.empty()){
        int u = q.front();
        q.pop();
        for (auto v : adj[u]) {
          if (vis[v] == -1) {
            vis[v] = 0;
            link[v] = u;
            dep[v] = dep[u] + 1;
            if (match[v] == -1) {
              for (int x = v, y = u, tmp;
                  y != -1; x = tmp, y =
                  x == -1 ? -1 : link[x
                ]) {
                tmp = match[y], match[x]
                    = y, match[y] = x;
              }
              return true;
            }
            q.push(match[v]), vis[match[v
                ]] = 1, dep[match[v]] =
                dep[u] + 2;
          } else if (vis[v] == 1 && find(
              v) != find(u)) {
            int p = lca(u, v);
            blossom(u, v, p), blossom(v,
                u, p);
          }
        }
      }
      return false;
    };
    int res = 0;
    for (int u = 0; u < n; ++u) { if (
        match[u] == -1) { res += augment
        (u); } }
    return res;
  }
};
```

2.7 Kuhn Munkres

```
// need perfect matching or not : w
    intialize with -INF / 0
template <typename Cost>
struct KM {
  static constexpr Cost INF =
      numeric_limits<Cost>::max() / 2;
  int n;
  vector<Cost> hl, hr, slk;
  vector<int> l, r, pre, vl, vr;
  queue<int> q;
  vector<vector<Cost>> w;
  KM(int n) : n(n), hl(n), hr(n), slk(n),
      l(n, -1), r(n, -1), pre(n), vl(n)
      , vr(n),
    w(n, vector<Cost>(n, -INF)) {}
  bool check(int x) {
    vl[x] = true;
    if (l[x] != -1) {
      q.push(l[x]);
      return vr[l[x]] = true;
    }
    while (x != -1) { swap(x, r[l[x] =
        pre[x]]); }
    return false;
  }
  void bfs(int s) {
    fill(slk.begin(), slk.end(), INF);
    fill(vl.begin(), vl.end(), false);
    fill(vr.begin(), vr.end(), false);
    q = {};
    q.push(s);
    vr[s] = true;
    while (true) {
      Cost d;
      while (!q.empty()) {
        int y = q.front();
        q.pop();
        for (int x = 0; x < n; ++x) {
          if (!vl[x] && slk[x] >= (d = hl
              [x] + hr[y] - w[x][y])) {
            pre[x] = y;
            if (d != 0) {
              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; }}
    }
  }
  void addEdge(int u, int v, Cost x) { w[
      u][v] = max(w[u][v], x); }
  Cost solve() {
    for (int i = 0; i < n; ++i) { hl[i] =
        *max_element(w[i].begin(), w[i
        ].end()); }
    for (int i = 0; i < n; ++i) { bfs(i);
        }
    Cost res = 0;
    for (int i = 0; i < n; ++i) { res +=
        w[i][l[i]]; }
    return res;
  }
};
```

2.8 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 ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
  1. Redirect every edge: $y \to x$ if $(x, y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x, y, c), connect $x \to y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost, cap) = (-c, 1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost, cap) = (0, d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost, cap) = (0, -d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u, v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.
- Project selection problem
  1. If $p_v > 0$, create edge (s, v) with capacity p_v; otherwise, create edge (v, t) with capacity $-p_v$.
  2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.
- 0/1 quadratic programming
  $$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x\bar{y} + x$$
  can be minimized by the mincut of the following graph:
  1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y.
  2. Create edge (x, y) with capacity c_{xy}.
  3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3 Data Structure

3.1 <ext/pbds>

```
#include <bits/extc++.h>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
#include <ext/pb_ds/assoc_container.hpp>
typedef tree<int, null_type, std::less<
    int>, rb_tree_tag,
    tree_order_statistics_node_update>
    tree_set;
typedef cc_hash_table<int, int> umap;
typedef priority_queue<int> heap;

int main() {
  // rb tree
  tree_set s;
  s.insert(71); s.insert(22);
  assert(*s.find_by_order(0) == 22);
      assert(*s.find_by_order(1) == 71);
  assert(s.order_of_key(22) == 0); assert
      (s.order_of_key(71) == 1);
  s.erase(22);
  assert(*s.find_by_order(0) == 71);
      assert(s.order_of_key(71) == 0);
  // mergable heap
```

```cpp
  heap a, b; a.join(b);
  // persistant
  rope<char> r[2];
  r[1] = r[0];
  std::string st = "abc";
  r[1].insert(0, st.c_str());
  r[1].erase(1, 1);
  std::cout << r[1].substr(0, 2) << std::
      endl;
  return 0;
}
```

3.2 Li Chao Tree

```cpp
constexpr i64 INF = 4e18;
struct Line {
  i64 a, b;
  Line() : a(0), b(INF) {}
  Line(i64 a, i64 b) : a(a), b(b) {}
  i64 operator()(i64 x) { return a * x +
      b; }
};
// [, ) !!!!!!!!!!!
struct Lichao {
  int n;
  vector<int> vals;
  vector<Line> lines;
  Lichao() {}
  void init(const vector<int> &v) {
    n = v.size();
    vals = v;
    sort(vals.begin(), vals.end());
    vals.erase(unique(vals.begin(), vals.
        end()), vals.end());
    lines.assign(4 * n, {});
  }
  int get(int x) { return lower_bound(
      vals.begin(), vals.end(), x) -
      vals.begin(); }
  void apply(Line p, int id, int l, int r
      ) {
    Line &q = lines[id];
    if (p(vals[l]) < q(vals[l])) { swap(p
        , q); }
    if (l + 1 == r) { return; }
    int m = l + r >> 1;
    if (p(vals[m]) < q(vals[m])) {
      swap(p, q);
      apply(p, id << 1, l, m);
    } else {
      apply(p, id << 1 | 1, m, r);
    }
  }
  void add(int ql, int qr, Line p) {
    ql = get(ql), qr = get(qr);
    auto go = [&](auto go, int id, int l,
        int r) -> void {
      if (qr <= l || r <= ql) { return; }
      if (ql <= l && r <= qr) {
        apply(p, id, l, r);
        return;
      }
      int m = l + r >> 1;
      go(go, id << 1, l, m);
      go(go, id << 1 | 1, m, r);
    };
    go(go, 1, 0, n);
  }
  i64 query(int p) {
    p = get(p);
    auto go = [&](auto go, int id, int l,
        int r) -> i64 {
      if (l + 1 == r) { return lines[id](
          vals[p]); }
      int m = l + r >> 1;
      return min(lines[id](vals[p]), p <
          m ? go(go, id << 1, l, m) : go
          (go, id << 1 | 1, m, r));
    };
    return go(go, 1, 0, n);
  }
};
```

3.3 Treap

```cpp
struct Treap {
    Treap *lc = nullptr, *rc = nullptr;
    int sz = 1;
    unsigned w = rng();
    i64 m = 0, b = 0, val = 0;
};
int size(Treap *t) {
    return t == nullptr ? 0 : t->sz;
}
void apply(Treap *t, i64 m, i64 b) {
    t->m += m;
    t->b += b;
    t->val += m * size(t->lc) + b;
}
void pull(Treap *t) {
    t->sz = size(t->lc) + size(t->rc) +
        1;
}
void push(Treap *t) {
    if (t->lc != nullptr) {
        apply(t->lc, t->m, t->b);
    }
    if (t->rc != nullptr) {
        apply(t->rc, t->m, t->b + t->m *
            (size(t->lc) + 1));
    }
    t->m = t->b = 0;
}
pair<Treap*, Treap*> split(Treap *t, int
    s) {
    if (t == nullptr) { return {t, t}; }
    push(t);
    Treap *a, *b;
    if (s <= size(t->lc)) {
        b = t;
        tie(a, b->lc) = split(t->lc, s);
    } else {
        a = t;
        tie(a->rc, b) = split(t->rc, s -
            size(t->lc) - 1);
    }
    pull(t);
    return {a, b};
}
Treap* merge(Treap *t1, Treap *t2) {
    if (t1 == nullptr) { return t2; }
    if (t2 == nullptr) { return t1; }
    push(t1), push(t2);
    if (t1->w > t2->w) {
        t1->rc = merge(t1->rc, t2);
        pull(t1);
        return t1;
    } else {
        t2->lc = merge(t1, t2->lc);
        pull(t2);
        return t2;
    }
}
int rnk(Treap *t, i64 val) {
    int res = 0;
    while (t != nullptr) {
        push(t);
        if (val <= t->val) {
            res += size(t->lc) + 1;
            t = t->rc;
        } else {
            t = t->lc;
        }
    }
    return res;
}
Treap* join(Treap *t1, Treap *t2) {
    if (size(t1) > size(t2)) {
        swap(t1, t2);
    }
    Treap *t = nullptr;
    while (t1 != nullptr) {
        auto [u1, v1] = split(t1, 1);
        t1 = v1;
        int r = rnk(t2, u1->val);
        if (r > 0) {
            auto [u2, v2] = split(t2, r);
            t = merge(t, u2);
            t2 = v2;
        }
        t = merge(t, u1);
    }
    t = merge(t, t2);
    return t;
}
```

3.4 Link-Cut Tree

```cpp
struct Splay {
  array<Splay*, 2> ch = {nullptr, nullptr
      };
  Splay* fa = nullptr;
  int sz = 1;
  bool rev = false;
  Splay() {}
  void applyRev(bool x) {
    if (x) {
      swap(ch[0], ch[1]);
      rev ^= 1;
    }
  }
  void push() {
    for (auto k : ch) {
      if (k) {
        k->applyRev(rev);
      }
    }
    rev = false;
  }
  void pull() {
    sz = 1;
    for (auto k : ch) {
      if (k) {
      }
    }
  }
  int relation() { return this == fa->ch
      [1]; }
  bool isRoot() { return !fa || fa->ch[0]
       != this && fa->ch[1] != this; }
  void rotate() {
    Splay *p = fa;
    bool x = !relation();
    p->ch[!x] = ch[x];
    if (ch[x]) { ch[x]->fa = p; }
    fa = p->fa;
    if (!p->isRoot()) { p->fa->ch[p->
        relation()] = this; }
    ch[x] = p;
    p->fa = this;
    p->pull();
  }
  void splay() {
    vector<Splay*> s;
    for (Splay *p = this; !p->isRoot(); p
        = p->fa) { s.push_back(p->fa);
        }
    while (!s.empty()) {
      s.back()->push();
      s.pop_back();
    }
    push();
    while (!isRoot()) {
      if (!fa->isRoot()) {
        if (relation() == fa->relation())
            {
          fa->rotate();
        } else {
          rotate();
        }
      }
      rotate();
    }
    pull();
  }
  void access() {
    for (Splay *p = this, *q = nullptr; p
        ; q = p, p = p->fa) {
      p->splay();
      p->ch[1] = q;
      p->pull();
    }
    splay();
  }
  void makeRoot() {
    access();
    applyRev(true);
  }
  Splay* findRoot() {
    access();
    Splay *p = this;
    while (p->ch[0]) { p = p->ch[0]; }
    p->splay();
```

```
    return p;
  }
  friend void split(Splay *x, Splay *y) {
    x->makeRoot();
    y->access();
  }
  // link if not connected
  friend void link(Splay *x, Splay *y) {
    x->makeRoot();
    if (y->findRoot() != x) {
      x->fa = y;
    }
  }
  // delete edge if doesn't exist
  friend void cut(Splay *x, Splay *y) {
    split(x, y);
    if (x->fa == y && !x->ch[1]) {
      x->fa = y->ch[0] = nullptr;
      x->pull();
    }
  }
  bool connected(Splay *x, Splay *y) {
    return x->findRoot() == y->findRoot()
        ;
  }
};
```

4 Graph

4.1 2-Edge-Connected Components

```
struct EBCC {
  int n, cnt = 0, T = 0;
  vector<vector<int>> adj, comps;
  vector<int> stk, dfn, low, id;
  EBCC(int n) : n(n), adj(n), dfn(n, -1),
       low(n), id(n, -1) {}
  void addEdge(int u, int v) { adj[u].
      push_back(v), adj[v].push_back(u);
       }
  void build() { for (int i = 0; i < n; i
      ++) { if (dfn[i] == -1) { dfs(i,
      -1); }}}
  void dfs(int u, int p) {
    dfn[u] = low[u] = T++;
    stk.push_back(u);
    for (auto v : adj[u]) {
      if (v == p) { continue; }
      if (dfn[v] == -1) {
        dfs(v, u);
        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 x;
      comps.emplace_back();
      do {
        x = stk.back();
        comps.back().push_back(x);
        id[x] = cnt;
        stk.pop_back();
      } while (x != u);
      cnt++;
    }
  }
};
```

4.2 2-Vertex-Connected Components

```
// is articulation point if appear in >=
   2 comps
auto dfs = [&](auto dfs, int u, int p) ->
     void {
  dfn[u] = low[u] = T++;
  for (auto v : adj[u]) {
    if (v == p) { continue; }
    if (dfn[v] == -1) {
      stk.push_back(v);
      dfs(dfs, v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= dfn[u]) {
        comps.emplace_back();
        int x;
        do {
          x = stk.back();
          cnt[x]++;
          stk.pop_back();
        } while (x != v);
        comps.back().push_back(u);
        cnt[u]++;
      }
    } else {
      low[u] = min(low[u], dfn[v]);
    }
  }
};
for (int i = 0; i < n; i++) {
  if (!adj[i].empty()) {
    dfs(dfs, i, -1);
  } else {
    comps.push_back({i});
  }
}
```

4.3 3-Edge-Connected Components

```
// DSU
struct ETCC {
  int n, cnt = 0;
  vector<vector<int>> adj, comps;
  vector<int> in, out, low, up, nx, id;
  ETCC(int n) : n(n), adj(n), in(n, -1),
       out(in), low(n), up(n), nx(in), id
       (in) {}
  void addEdge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  void build() {
    int T = 0;
    DSU d(n);
    auto merge = [&](int u, int v) {
      d.join(u, v);
      up[u] += up[v];
    };
    auto dfs = [&](auto dfs, int u, int p
        ) -> void {
      in[u] = low[u] = T++;
      for (auto v : adj[u]) {
        if (v == u) { continue; }
        if (v == p) {
          p = -1;
          continue;
        }
        if (in[v] == -1) {
          dfs(dfs, v, u);
          if (nx[v] == -1 && up[v] <= 1)
              {
            up[u] += up[v];
            low[u] = min(low[u], low[v]);
            continue;
          }
          if (up[v] == 0) { v = nx[v]; }
          if (low[u] > low[v]) { low[u] =
               low[v], swap(nx[u], v); }
          while (v != -1) { merge(u, v);
               v = nx[v]; }
        } else if (in[v] < in[u]) {
          low[u] = min(low[u], in[v]);
          up[u]++;
        } else {
          for (int &x = nx[u]; x != -1 &&
               in[x] <= in[v] && in[v] <
               out[x]; x = nx[x]) {
            merge(u, x);
          }
          up[u]--;
        }
      }
      out[u] = T;
    };
    for (int i = 0; i < n; i++) { if (in[
        i] == -1) { dfs(dfs, i, -1); }}
    for (int i = 0; i < n; i++) { if (d.
        find(i) == i) { id[i] = cnt++;
        }}
    comps.resize(cnt);
    for (int i = 0; i < n; i++) { comps[
        id[d.find(i)]].push_back(i); }
  }
};
```

4.4 Heavy-Light Decomposition

```
struct HLD {
  int n, cur = 0;
  vector<int> sz, top, dep, par, tin,
       tout, seq;
  vector<vector<int>> adj;
  HLD(int n) : n(n), sz(n, 1), top(n),
       dep(n), par(n), tin(n), tout(n),
       seq(n), adj(n) {}
  void addEdge(int u, int v) { adj[u].
       push_back(v), adj[v].push_back(u);
        }
  void build(int root = 0) {
    top[root] = root, dep[root] = 0, par[
         root] = -1;
    dfs1(root), dfs2(root);
  }
  void dfs1(int u) {
    if (auto it = find(adj[u].begin(),
         adj[u].end(), par[u]); it != adj
         [u].end()) {
      adj[u].erase(it);
    }
    for (auto &v : adj[u]) {
      par[v] = u;
      dep[v] = dep[u] + 1;
      dfs1(v);
      sz[u] += sz[v];
      if (sz[v] > sz[adj[u][0]]) { swap(v
          , adj[u][0]); }
    }
  }
  void dfs2(int u) {
    tin[u] = cur++;
    seq[tin[u]] = u;
    for (auto v : adj[u]) {
      top[v] = v == adj[u][0] ? top[u] :
           v;
      dfs2(v);
    }
    tout[u] = cur - 1;
  }
  int lca(int u, int v) {
    while (top[u] != top[v]) {
      if (dep[top[u]] > dep[top[v]]) {
        u = par[top[u]];
      } else {
        v = par[top[v]];
      }
    }
    return dep[u] < dep[v] ? u : v;
  }
  int dist(int u, int v) { return dep[u]
       + dep[v] - 2 * dep[lca(u, v)]; }
  int jump(int u, int k) {
    if (dep[u] < k) { return -1; }
    int d = dep[u] - k;
    while (dep[top[u]] > d) { u = par[top
        [u]]; }
    return seq[tin[u] - dep[u] + d];
  }
  // u is v's ancestor
  bool isAncestor(int u, int v) { return
       tin[u] <= tin[v] && tin[v] <= tout
       [u]; }
  // root's parent is itself
  int rootedParent(int r, int u) {
    if (r == u) { return u; }
    if (isAncestor(r, u)) { return par[u
        ]; }
    auto it = upper_bound(adj[u].begin(),
         adj[u].end(), r, [&](int x, int
         y) {
      return tin[x] < tin[y];
    }) - 1;
    return *it;
  }
```

```cpp
  // rooted at u, v's subtree size
  int rootedSize(int r, int u) {
    if (r == u) { return n; }
    if (isAncestor(u, r)) { return sz[u];
      }
    return n - sz[rootedParent(r, u)];
  }
  int rootedLca(int r, int a, int b) {
      return lca(a, b) ^ lca(a, r) ^ lca
      (b, r); }
};
```

4.5 Centroid Decomposition

```cpp
vector<int> sz(n), vis(n);
auto build = [&](auto build, int u, int p
    ) -> void {
  sz[u] = 1;
  for (auto v : g[u]) {
    if (v != p && !vis[v]) {
      build(build, v, u);
      sz[u] += sz[v];
    }
  }
};
auto find = [&](auto find, int u, int p,
    int tot) -> int {
  for (auto v : g[u]) {
    if (v != p && !vis[v] && 2 * sz[v] >
        tot) {
      return find(find, v, u, tot);
    }
  }
  return u;
};

auto dfs = [&](auto dfs, int cen) -> void
    {
  build(build, cen, -1);
  cen = find(find, cen, -1, sz[cen]);
  vis[cen] = 1;
  build(build, cen, -1);

  for (auto v : g[cen]) {
    if (!vis[v]) {
      dfs(dfs, v);
    }
  }
};
dfs(dfs, 0);
```

4.6 Strongly Connected Components

```cpp
struct SCC {
  int n, cnt = 0, cur = 0;
  vector<int> id, dfn, low, stk;
  vector<vector<int>> adj, comps;
  void addEdge(int u, int v) { adj[u].
      push_back(v); }
  SCC(int n) : n(n), id(n, -1), dfn(n,
      -1), low(n, -1), adj(n) {}
  void build() {
    auto dfs = [&](auto dfs, int u) ->
        void {
      dfn[u] = low[u] = cur++;
      stk.push_back(u);
      for (auto v : adj[u]) {
        if (dfn[v] == -1) {
          dfs(dfs, 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;
        comps.emplace_back();
        do {
          v = stk.back();
          comps.back().push_back(v);
          id[v] = cnt;
          stk.pop_back();
        } while (u != v);
        cnt++;
      }
    };
    for (int i = 0; i < n; i++) { if (dfn
        [i] == -1) { dfs(dfs, i); }}
    for (int i = 0; i < n; i++) { id[i] =
        cnt - 1 - id[i]; }
    reverse(comps.begin(), comps.end());
  }
  // the comps are in topological sorted
      order
};
```

4.7 2-SAT

```cpp
struct TwoSat {
  int n, N;
  vector<vector<int>> adj;
  vector<int> ans;
  TwoSat(int n) : n(n), N(n), adj(2 * n)
      {}
  // u == x
  void addClause(int u, bool x) { adj[2 *
       u + !x].push_back(2 * u + x); }
  // u == x || v == y
  void addClause(int u, bool x, int v,
      bool y) {
    adj[2 * u + !x].push_back(2 * v + y);
    adj[2 * v + !y].push_back(2 * u + x);
  }
  // u == x -> v == y
  void addImply(int u, bool x, int v,
      bool y) { addClause(u, !x, v, y);
      }
  void addVar() {
    adj.emplace_back(), adj.emplace_back
        ();
    N++;
  }
  // at most one in var is true
  // adds prefix or as supplementary
      variables
  void atMostOne(const vector<pair<int,
      bool>> &vars) {
    int sz = vars.size();
    for (int i = 0; i < sz; i++) {
      addVar();
      auto [u, x] = vars[i];
      addImply(u, x, N - 1, true);
      if (i > 0) {
        addImply(N - 2, true, N - 1, true
            );
        addClause(u, !x, N - 2, false);
      }
    }
  }
  // does not return supplementary
      variables from atMostOne()
  bool satisfiable() {
    // run tarjan scc on 2 * N
    for (int i = 0; i < 2 * N; i++) { if
        (dfn[i] == -1) { dfs(dfs, i); }}
    for (int i = 0; i < N; i++) { if (id
        [2 * i] == id[2 * i + 1]) {
        return false; }}
    ans.resize(n);
    for (int i = 0; i < n; i++) { ans[i]
        = id[2 * i] > id[2 * i + 1]; }
    return true;
  }
};
```

4.8 count 3-cycles and 4-cycles

```cpp
sort(ord.begin(), ord.end(), [&](auto i,
    auto j) { return pair(deg[i], i) >
    pair(deg[j], j); });
for (int i = 0; i < n; i++) { rnk[ord[i]]
     = i; }
if (rnk[u] < rnk[v]) { dag[u].push_back(v
    ); }
// c3
for (int x = 0; x < n; x++) {
  for (auto y : dag[x]) { vis[y] = 1; }
  for (auto y : dag[x]) { for (auto z :
      dag[y]) { ans += vis[z]; }}
  for (auto y : dag[x]) { vis[y] = 0; }
}
// c4
for (int x = 0; x < n; x++) {
  for (auto y : dag[x]) { for (auto z :
      adj[y]) { if (rnk[z] > rnk[x]) {
      ans += vis[z]++; }}}
  for (auto y : dag[x]) { for (auto z :
      adj[y]) { if (rnk[z] > rnk[x]) {
      vis[z]--; }}}
}
```

4.9 Minimum Mean Cycle

create a new vertex S, connect S to all vertices with arbitrary weight (0). Let $f_i(u)$ be the shortest path from S to u with exactly i edges.

$$ans = \min_{f_{n+1}(i)!=\infty} \max_{j=1}^{n} \frac{f_{n+1}(i) - f_j(i)}{n+1-j}$$

4.10 Directed Minimum Spanning Tree

```cpp
// DSU with rollback
template <typename Cost>
struct DMST {
  int n;
  vector<int> s, t, lc, rc, h;
  vector<Cost> c, tag;
  DMST(int n) : n(n), h(n, -1) {}
  void addEdge(int u, int v, Cost w) {
    int id = s.size();
    s.push_back(u), t.push_back(v), c.
        push_back(w);
    lc.push_back(-1), rc.push_back(-1);
    tag.emplace_back();
    h[v] = merge(h[v], id);
  }
  pair<Cost, vector<int>> build(int root
       = 0) {
    DSU d(n);
    Cost res{};
    vector<int> vis(n, -1), path(n), q(n)
        , in(n, -1);
    vis[root] = root;
    vector<pair<int, vector<int>>> cycles
        ;
    for (auto r = 0; r < n; ++r) {
      auto u = r, b = 0, w = -1;
      while (!~vis[u]) {
        if (!~h[u]) { return {-1, {}}; }
        push(h[u]);
        int e = h[u];
        res += c[e], tag[h[u]] -= c[e];
        h[u] = pop(h[u]);
        q[b] = e, path[b++] = u, vis[u] =
             r;
        u = d.find(s[e]);
        if (vis[u] == r) {
          int cycle = -1, e = b;
          do {
            w = path[--b];
            cycle = merge(cycle, h[w]);
          } while (d.join(u, w));
          u = d.find(u);
          h[u] = cycle, vis[u] = -1;
          cycles.emplace_back(u, vector<
              int>(q.begin() + b, q.
              begin() + e));
        }
      }
      for (auto i = 0; i < b; ++i) { in[d
          .find(t[q[i]])] = q[i]; }
    }
    reverse(cycles.begin(), cycles.end())
        ;
    for (const auto &[u, comp] : cycles)
        {
      int count = int(comp.size()) - 1;
      d.back(count);
      int ine = in[u];
      for (auto e : comp) { in[d.find(t[e
          ]])] = e; }
      in[d.find(t[ine])] = ine;
    }
    vector<int> par;
```

```cpp
    par.reserve(n);
    for (auto i : in) { par.push_back(i != -1 ? s[i] : -1); }
    return {res, par};
  }
  void push(int u) {
    c[u] += tag[u];
    if (int l = lc[u]; l != -1) { tag[l] += tag[u]; }
    if (int r = rc[u]; r != -1) { tag[r] += tag[u]; }
    tag[u] = 0;
  }
  int merge(int u, int v) {
    if (u == -1 || v == -1) { return u != -1 ? u : v; }
    push(u);
    push(v);
    if (c[u] > c[v]) { swap(u, v); }
    rc[u] = merge(v, rc[u]);
    swap(lc[u], rc[u]);
    return u;
  }
  int pop(int u) {
    push(u);
    return merge(lc[u], rc[u]);
  }
};
```

4.11 Maximum Clique

```cpp
pair<int, vector<int>> maxClique(int n, const vector<bitset<N>> adj) {
  int mx = 0;
  vector<int> ans, cur;
  auto rec = [&](auto rec, bitset<N> s) -> void {
    int sz = s.count();
    if (int(cur.size()) > mx) { mx = cur.size(), ans = cur; }
    if (int(cur.size()) + sz <= mx) { return; }
    int e1 = -1, e2 = -1;
    vector<int> d(n);
    for (int i = 0; i < n; i++) {
      if (s[i]) {
        d[i] = (adj[i] & s).count();
        if (e1 == -1 || d[i] > d[e1]) { e1 = i; }
        if (e2 == -1 || d[i] < d[e2]) { e2 = i; }
      }
    }
    if (d[e1] >= sz - 2) {
      cur.push_back(e1);
      auto s1 = adj[e1] & s;
      rec(rec, s1);
      cur.pop_back();
      return;
    }
    cur.push_back(e2);
    auto s2 = adj[e2] & s;
    rec(rec, s2);
    cur.pop_back();
    s.reset(e2);
    rec(rec, s);
  };
  bitset<N> all;
  for (int i = 0; i < n; i++) {
    all.set(i);
  }
  rec(rec, all);
  return pair(mx, ans);
}
```

4.12 Dominator Tree

```cpp
// res : parent of each vertex in dominator tree, -1 is root, -2 if not in tree
struct DominatorTree {
  int n, cur = 0;
  vector<int> dfn, rev, fa, sdom, dom, val, rp, res;
  vector<vector<int>> adj, rdom, r;
  DominatorTree(int n) : n(n), dfn(n, -1), res(n, -2), adj(n), rdom(n), r(n) {
    rev = fa = sdom = dom = val = rp = dfn;
  }
  void addEdge(int u, int v) {
    adj[u].push_back(v);
  }
  void dfs(int u) {
    dfn[u] = cur;
    rev[cur] = u;
    fa[cur] = sdom[cur] = val[cur] = cur;
    cur++;
    for (int v : adj[u]) {
      if (dfn[v] == -1) {
        dfs(v);
        rp[dfn[v]] = dfn[u];
      }
      r[dfn[v]].push_back(dfn[u]);
    }
  }
  int find(int u, int c) {
    if (fa[u] == u) { return c != 0 ? -1 : u; }
    int p = find(fa[u], 1);
    if (p == -1) { return c != 0 ? fa[u] : val[u]; }
    if (sdom[val[u]] > sdom[val[fa[u]]]) { val[u] = val[fa[u]]; }
    fa[u] = p;
    return c != 0 ? p : val[u];
  }
  void build(int s = 0) {
    dfs(s);
    for (int i = cur - 1; i >= 0; i--) {
      for (int u : r[i]) { sdom[i] = min(sdom[i], sdom[find(u, 0)]); }
      if (i > 0) { rdom[sdom[i]].push_back(i); }
      for (int u : rdom[i]) {
        int p = find(u, 0);
        if (sdom[p] == i) {
          dom[u] = i;
        } else {
          dom[u] = p;
        }
      }
      if (i > 0) { fa[i] = rp[i]; }
    }
    res[s] = -1;
    for (int i = 1; i < cur; i++) { if (sdom[i] != dom[i]) { dom[i] = dom[dom[i]]; }}
    for (int i = 1; i < cur; i++) { res[rev[i]] = rev[dom[i]]; }
  }
};
```

4.13 Edge Coloring

```cpp
// bipartite
e[i] = pair(u, v + a), deg[u]++, deg[v + a]++;
int col = *max_element(deg.begin(), deg.end());
vector<int> ans(m, -1);
vector has(a + b, vector<pair<int, int>>(col, {-1, -1}));
for (int i = 0; i < m; i++) {
  auto [u, v] = e[i];
  vector<int> c;
  for (auto x : {u, v}) {
    c.push_back(0);
    while (has[x][c.back()].first != -1) { c.back()++; }
  }
  if (c[0] != c[1]) {
    auto dfs = [&](auto dfs, int u, int x) -> void {
      auto [v, i] = has[u][c[x]];
      if (v != -1) {
        if (has[v][c[x ^ 1]].first != -1) {
          dfs(dfs, v, x ^ 1);
        } else {
          has[v][c[x]] = {-1, -1};
        }
        has[u][c[x ^ 1]] = {v, i}, has[v][c[x ^ 1]] = {u, i};
        ans[i] = c[x ^ 1];
      }
    };
    dfs(dfs, v, 0);
  }
  has[u][c[0]] = {v, i};
  has[v][c[0]] = {u, i};
  ans[i] = c[0];
}
// general
auto vizing(int n, const vector<pair<int, int>> &e) {
  vector<int> deg(n);
  for (auto [u, v] : e) {
    deg[u]++, deg[v]++;
  }
  int col = *max_element(deg.begin(), deg.end()) + 1;
  vector<int> free(n);
  vector ans(n, vector<int>(n, -1));
  vector at(n, vector<int>(col, -1));
  auto update = [&](int u) {
    free[u] = 0;
    while (at[u][free[u]] != -1) {
      free[u]++;
    }
  };
  auto color = [&](int u, int v, int c1) {
    int c2 = ans[u][v];
    ans[u][v] = ans[v][u] = c1;
    at[u][c1] = v, at[v][c1] = u;
    if (c2 != -1) {
      at[u][c2] = at[v][c2] = -1;
      free[u] = free[v] = c2;
    } else {
      update(u), update(v);
    }
    return c2;
  };
  auto flip = [&](int u, int c1, int c2) {
    int v = at[u][c1];
    swap(at[u][c1], at[u][c2]);
    if (v != -1) {
      ans[u][v] = ans[v][u] = c2;
    }
    if (at[u][c1] == -1) {
      free[u] = c1;
    }
    if (at[u][c2] == -1) {
      free[u] = c2;
    }
    return v;
  };
  for (int i = 0; i < int(e.size()); i++) {
    auto [u, v1] = e[i];
    int v2 = v1, c1 = free[u], c2 = c1, d;
    vector<pair<int, int>> fan;
    vector<int> vis(col);
    while (ans[u][v1] == -1) {
      fan.emplace_back(v2, d = free[v2]);
      if (at[v2][c2] == -1) {
        for (int j = int(fan.size()) - 1; j >= 0; j--) {
          c2 = color(u, fan[j].first, c2);
        }
      } else if (at[u][d] == -1) {
        for (int j = int(fan.size()) - 1; j >= 0; j--) {
          color(u, fan[j].first, fan[j].second);
        }
      } else if (vis[d] == 1) {
        break;
      } else {
        vis[d] = 1, v2 = at[u][d];
      }
    }
    if (ans[u][v1] == -1) {
```

```
      while (v2 != -1) {
        v2= flip(v2, c2, d);
        swap(c2, d);
      }
      if (at[u][c1] != -1) {
        int j = int(fan.size()) - 2;
        while (j >= 0 && fan[j].second != c2) {
          j--;
        }
        while (j >= 0) {
          color(u, fan[j].first, fan[j].second);
          j--;
        }
      } else {
        i--;
      }
    }
  }
  return pair(col, ans);
}
```

5 String

5.1 Prefix Function

```
template <typename T>
vector<int> prefixFunction(const T &s) {
  int n = int(s.size());
  vector<int> p(n);
  for (int i = 1; i < n; i++) {
    int j = p[i - 1];
    while (j > 0 && s[i] != s[j]) { j = p[j - 1]; }
    if (s[i] == s[j]) { j++; }
    p[i] = j;
  }
  return p;
}
```

5.2 Z Function

```
template <typename T>
vector<int> zFunction(const T &s) {
  int n = int(s.size());
  if (n == 0) return {};
  vector<int> z(n);
  for (int i = 1, j = 0; i < n; i++) {
    int &k = z[i];
    k = j + z[j] <= i ? 0 : min(j + z[j] - i, z[i - j]);
    while (i + k < n && s[k] == s[i + k]) { k++; }
    if (j + z[j] < i + z[i]) { j = i; }
  }
  z[0] = n;
  return z;
}
```

5.3 Suffix Array

```
// need to discretize
struct SuffixArray {
  int n;
  vector<int> sa, as, ha;
template <typename T>
  vector<int> sais(const T &s) {
    int n = s.size(), m = *max_element(s.begin(), s.end()) + 1;
    vector<int> pos(m + 1), f(n);
    for (auto ch : s) { pos[ch + 1]++; }
    for (int i = 0; i < m; i++) { pos[i + 1] += pos[i]; }
    for (int i = n - 2; i >= 0; i--) { f[i] = s[i] != s[i + 1] ? s[i] < s[i + 1] : f[i + 1]; }
    vector<int> x(m), sa(n);
    auto induce = [&](const vector<int> &ls) {
      fill(sa.begin(), sa.end(), -1);
      auto L = [&](int i) { if (i >= 0 && !f[i]) { sa[x[s[i]]++] = i; }};
      auto S = [&](int i) { if (i >= 0 && f[i]) { sa[--x[s[i]]] = i; }};
      for (int i = 0; i < m; i++) { x[i] = pos[i + 1]; }
      for (int i = int(ls.size()) - 1; i >= 0; i--) { S(ls[i]); }
      for (int i = 0; i < m; i++) { x[i] = pos[i]; }
      L(n - 1);
      for (int i = 0; i < n; i++) { L(sa[i] - 1); }
      for (int i = 0; i < m; i++) { x[i] = pos[i + 1]; }
      for (int i = n - 1; i >= 0; i--) { S(sa[i] - 1); }
    };
    auto ok = [&](int i) { return i == n || !f[i - 1] && f[i]; };
    auto same = [&](int i, int j) {
      do { if (s[i++] != s[j++]) { return false; }} while (!ok(i) && !ok(j));
      return ok(i) && ok(j);
    };
    vector<int> val(n), lms;
    for (int i = 1; i < n; i++) { if (ok(i)) { lms.push_back(i); }}
    induce(lms);
    if (!lms.empty()) {
      int p = -1, w = 0;
      for (auto v : sa) {
        if (v != 0 && ok(v)) {
          if (p != -1 && same(p, v)) { w--; }
          val[p = v] = w++;
        }
      }
      auto b = lms;
      for (auto &v : b) { v = val[v]; }
      b = sais(b);
      for (auto &v : b) { v = lms[v]; }
      induce(b);
    }
    return sa;
  }
template <typename T>
  SuffixArray(const T &s) : n(s.size()), sa(sais(s)), as(n), ha(n - 1) {
    for (int i = 0; i < n; i++) { as[sa[i]] = i; }
    for (int i = 0, j = 0; i < n; ++i) {
      if (as[i] == 0) {
        j = 0;
      } else {
        for (j -= j > 0; i + j < n && sa[as[i] - 1] + j < n && s[i + j] == s[sa[as[i] - 1] + j]; ) { ++j; }
        ha[as[i] - 1] = j;
      }
    }
  }
};
```

5.4 Manacher's Algorithm

```
// returns radius of t, length of s : rad(t) - 1, radius of s : rad(t) / 2
vector<int> manacher(string s) {
  string t = "#";
  for (auto c : s) { t += c, t += '#'; }
  int n = t.size();
  vector<int> r(n);
  for (int i = 0, j = 0; i < n; i++) {
    if (2 * j - i >= 0 && j + r[j] > i) { r[i] = min(r[2 * j - i], j + r[j] - i); }
    while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) { r[i]++; }
    if (i + r[i] > j + r[j]) { j = i; }
  }
  return r;
}
```

5.5 Aho-Corasick Automaton

```
constexpr int K = 26;
struct Node {
  array<int, K> nxt;
  int fail = -1;
  // other vars
  Node() { nxt.fill(-1); }
};
vector<Node> aho(1);
for (int i = 0; i < n; i++) {
  string s;
  cin >> s;
  int u = 0;
  for (auto ch : s) {
    int c = ch - 'a';
    if (aho[u].nxt[c] == -1) {
      aho[u].nxt[c] = aho.size();
      aho.emplace_back();
    }
    u = aho[u].nxt[c];
  }
}
vector<int> q;
for (auto &i : aho[0].nxt) {
  if (i == -1) {
    i = 0;
  } else {
    q.push_back(i);
    aho[i].fail = 0;
  }
}
for (int i = 0; i < int(q.size()); i++) {
  int u = q[i];
  if (u > 0) {
    // maintain
  }
  for (int c = 0; c < K; c++) {
    if (int v = aho[u].nxt[c]; v != -1) {
      aho[v].fail = aho[aho[u].fail].nxt[c];
      q.push_back(v);
    } else {
      aho[u].nxt[c] = aho[aho[u].fail].nxt[c];
    }
  }
}
```

5.6 Suffix Automaton

```
struct SAM {
  static constexpr int A = 26;
  struct Node {
    int len = 0, link = -1, cnt = 0;
    array<int, A> nxt;
    Node() { nxt.fill(-1); }
  };
  vector<Node> t;
  SAM() : t(1) {}
  int size() { return t.size(); }
  Node& operator[](int i) { return t[i]; }
  int newNode() {
    t.emplace_back();
    return t.size() - 1;
  }
  int extend(int p, int c) {
    int cur = newNode();
    t[cur].len = t[p].len + 1;
    t[cur].cnt = 1;
    while (p != -1 && t[p].nxt[c] == -1) {
      t[p].nxt[c] = cur;
      p = t[p].link;
    }
    if (p == -1) {
      t[cur].link = 0;
    } else {
      int q = t[p].nxt[c];
      if (t[p].len + 1 == t[q].len) {
        t[cur].link = q;
      } else {
        int clone = newNode();
        t[clone].len = t[p].len + 1;
```

```
      t[clone].link = t[q].link;
      t[clone].nxt = t[q].nxt;
      while (p != -1 && t[p].nxt[c] ==
          q) {
        t[p].nxt[c] = clone;
        p = t[p].link;
      }
      t[q].link = t[cur].link = clone;
    }
  }
  return cur;
}
};
```

5.7 Lexicographically Smallest Rotation

```
template <typename T>
T minRotation(T s) {
  int n = s.size();
  int i = 0, j = 1;
  s.insert(s.end(), s.begin(), s.end());
  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 T(s.begin() + ans, s.begin() +
      ans + n);
}
```

5.8 EER Tree

```
// cnt : occurrences, (dfs fail tree)
// num : number of pal ending here
struct PAM {
  static constexpr int A = 26;
  struct Node {
    int len = 0, link = 0, cnt = 0, num =
        0;
    array<int, A> nxt{};
    Node() {}
  };
  vector<Node> t;
  int suf = 1;
  string s;
  PAM() : t(2) { t[0].len = -1; }
  int size() { return t.size(); }
  Node& operator[](int i) { return t[i];
      }
  int newNode() {
    t.emplace_back();
    return t.size() - 1;
  }
  bool add(int c, char offset = 'a') {
    int pos = s.size();
    s += c + offset;
    int cur = suf, curlen = 0;
    while (true) {
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos
          - 1 - curlen] == s[pos]) {
        break; }
      cur = t[cur].link;
    }
    if (t[cur].nxt[c]) {
      suf = t[cur].nxt[c];
      t[suf].cnt++;
      return false;
    }
    suf = newNode();
    t[suf].len = t[cur].len + 2;
    t[suf].cnt = t[suf].num = 1;
    t[cur].nxt[c] = suf;
    if (t[suf].len == 1) {
      t[suf].link = 1;
      return true;
    }
    while (true) {
      cur = t[cur].link;
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos
          - 1 - curlen] == s[pos]) {
        t[suf].link = t[cur].nxt[c];
        break;
      }
    }
    t[suf].num += t[t[suf].link].num;
    return true;
  }
};
```

6 Math

6.1 Extended GCD

```
array<i64, 3> extgcd(i64 a, i64 b) {
  if (b == 0) { return {a, 1, 0}; }
  auto [g, x, y] = extgcd(b, a % b);
  return {g, y, x - a / b * y};
}
```

6.2 Chinese Remainder Theorem

```
// returns (rem, mod), n = 0 return (0,
    1), no solution return (0, 0)
pair<i64, i64> crt(vector<i64> r, vector<
    i64> m) {
  int n = r.size();
  for (int i = 0; i < n; i++) {
    r[i] %= m[i];
    if (r[i] < 0) { r[i] += m[i]; }
  }
  i64 r0 = 0, m0 = 1;
  for (int i = 0; i < n; i++) {
    i64 r1 = r[i], m1 = m[i];
    if (m0 < m1) { swap(r0, r1), swap(m0,
        m1); }
    if (m0 % m1 == 0) {
      if (r0 % m1 != r1) { return {0, 0};
          }
      continue;
    }
    auto [g, a, b] = extgcd(m0, m1);
    i64 u1 = m1 / g;
    if ((r1 - r0) % g != 0) { return {0,
        0}; }
    i64 x = (r1 - r0) / g % u1 * a % u1;
    r0 += x * m0;
    m0 *= u1;
    if (r0 < 0) { r0 += m0; }
  }
  return {r0, m0};
}
```

6.3 NTT and polynomials

```
template <int P>
struct Modint {
  int v;
  // need constexpr, constructor, +-*,
      qpow, inv()
};
template<int P>
constexpr Modint<P> findPrimitiveRoot() {
  Modint<P> i = 2;
  int k = __builtin_ctz(P - 1);
  while (true) {
    if (i.qpow((P - 1) / 2).v != 1) {
        break; }
    i = i + 1;
  }
  return i.qpow(P - 1 >> k);
}
template <int P>
constexpr Modint<P> primitiveRoot =
    findPrimitiveRoot<P>();
vector<int> rev;
template <int P>
vector<Modint<P>> roots{0, 1};
template <int P>
void dft(vector<Modint<P>> &a) {
  int n = a.size();
  if (n == 1) { return; }
  if (int(rev.size()) != n) {
    int k = __builtin_ctz(n) - 1;
    rev.resize(n);
    for (int i = 0; i < n; i++) { rev[i]
        = rev[i >> 1] >> 1 | (i & 1) <<
        k; }
  }
  for (int i = 0; i < n; i++) { if (rev[i
      ] < i) { swap(a[i], a[rev[i]]); }}
  if (roots<P>.size() < n) {
    int k = __builtin_ctz(roots<P>.size()
        );
    roots<P>.resize(n);
    while ((1 << k) < n) {
      auto e = Modint<P>(primitiveRoot<P
          >).qpow(P - 1 >> k + 1);
      for (int i = 1 << k - 1; i < 1 << k
          ; i++) {
        roots<P>[2 * i] = roots<P>[i];
        roots<P>[2 * i + 1] = roots<P>[i]
            * e;
      }
      k++;
    }
  }
  // fft : just do roots[i] = exp(2 * PI
      / n * i * complex<double>(0, 1))
  for (int k = 1; k < n; k *= 2) {
    for (int i = 0; i < n; i += 2 * k) {
      for (int j = 0; j < k; j++) {
        Modint<P> u = a[i + j];
        Modint<P> v = a[i + j + k] *
            roots<P>[k + j];
        // fft : v = a[i + j + k] * roots
            [n / (2 * k) * j]
        a[i + j] = u + v;
        a[i + j + k] = u - v;
      }
    }
  }
}
template <int P>
void idft(vector<Modint<P>> &a) {
  int n = a.size();
  reverse(a.begin() + 1, a.end());
  dft(a);
  Modint<P> x = (1 - P) / n;
  for (int i = 0; i < n; i++) { a[i] = a[
      i] * x; }
}
template <int P>
struct Poly : vector<Modint<P>> {
  using Mint = Modint<P>;
  Poly() {}
  explicit Poly(int n) : vector<Mint>(n)
      {}
  explicit Poly(const vector<Mint> &a) :
      vector<Mint>(a) {}
  explicit Poly(const initializer_list<
      Mint> &a) : vector<Mint>(a) {}
template<class F>
  explicit Poly(int n, F f) : vector<Mint
      >(n) { for (int i = 0; i < n; i++)
        { (*this)[i] = f(i); }}
template<class InputIt>
  explicit constexpr Poly(InputIt first,
      InputIt last) : vector<Mint>(first
      , last) {}
  Poly mulxk(int k) {
    auto b = *this;
    b.insert(b.begin(), k, 0);
    return b;
  }
  Poly modxk(int k) {
    k = min(k, int(this->size()));
    return Poly(this->begin(), this->
        begin() + k);
  }
  Poly divxk(int k) {
    if (this->size() <= k) { return Poly
        (); }
```

```cpp
    return Poly(this->begin() + k, this->end());
  }
  friend Poly operator+(const Poly &a, const Poly &b) {
    Poly res(max(a.size(), b.size()));
    for (int i = 0; i < int(a.size()); i++) { res[i] = res[i] + a[i]; }
    for (int i = 0; i < int(b.size()); i++) { res[i] = res[i] + b[i]; }
    return res;
  }
  friend Poly operator-(const Poly &a, const Poly &b) {
    Poly res(max(a.size(), b.size()));
    for (int i = 0; i < int(a.size()); i++) { res[i] = res[i] + a[i]; }
    for (int i = 0; i < int(b.size()); i++) { res[i] = res[i] - b[i]; }
    return res;
  }
  friend Poly operator*(Poly a, Poly b) {
    if (a.empty() || b.empty()) { return Poly(); }
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot) { sz *= 2; }
    a.resize(sz);
    b.resize(sz);
    dft(a);
    dft(b);
    for (int i = 0; i < sz; i++) { a[i] = a[i] * b[i]; }
    idft(a);
    a.resize(tot);
    return a;
  }
  friend Poly operator*(Poly a, Mint b) {
    for (int i = 0; i < int(a.size()); i++) { a[i] = a[i] * b; }
    return a;
  }
  Poly derivative()  {
    if (this->empty()) { return Poly(); }
    Poly res(this->size() - 1);
    for (int i = 0; i < this->size() - 1; ++i) { res[i] = (i + 1) * (*this)[i + 1]; }
    return res;
  }
  Poly integral() {
    Poly res(this->size() + 1);
    for (int i = 0; i < this->size(); ++i) { res[i + 1] = (*this)[i] * Mint(i + 1).inv(); }
    return res;
  }
  Poly inv(int m) {
    // a[0] != 0
    Poly x({(*this)[0].inv()});
    int k = 1;
    while (k < m) {
      k *= 2;
      x = (x * (Poly({2}) - modxk(k) * x)).modxk(k);
    }
    return x.modxk(m);
  }
  Poly log(int m)  {
    return (derivative() * inv(m)).integral().modxk(m);
  }
  Poly exp(int m) {
    Poly x({1});
    int k = 1;
    while (k < m) {
      k *= 2;
      x = (x * (Poly({1}) - x.log(k) + modxk(k))).modxk(k);
    }
    return x.modxk(m);
  }
  Poly pow(i64 k, int m) {
    if (k == 0) { return Poly(m, [&](int i) { return i == 0; }); }
    int i = 0;
    while (i < this->size() && (*this)[i].v == 0) { i++; }
    if (i == this->size() || __int128(i) * k >= m) { return Poly(m); }
    Mint v = (*this)[i];
    auto f = divxk(i) * v.inv();
    return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * v.qpow(k);
  }
  Poly sqrt(int m) {
    // a[0] == 1, otherwise quadratic residue?
    Poly x({1});
    int k = 1;
    while (k < m) {
      k *= 2;
      x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
    }
    return x.modxk(m);
  }
  Poly mulT(Poly b) const {
    if (b.empty()) { return Poly(); }
    int n = b.size();
    reverse(b.begin(), b.end());
    return (*this * b).divxk(n - 1);
  }
  vector<Mint> evaluate(vector<Mint> x) {
    if (this->empty()) { return vector<Mint>(x.size()); }
    int n = max(x.size(), this->size());
    vector<Poly> q(4 * n);
    vector<Mint> ans(x.size());
    x.resize(n);
    auto build = [&](auto build, int id, int l, int r) -> void {
      if (r - l == 1) {
        q[id] = Poly({1, -x[l].v});
      } else {
        int m = (l + r) / 2;
        build(build, 2 * id, l, m);
        build(build, 2 * id + 1, m, r);
        q[id] = q[2 * id] * q[2 * id + 1];
      }
    };
    build(build, 1, 0, n);
    auto work = [&](auto work, int id, int l, int r, const Poly &num) -> void {
      if (r - l == 1) {
        if (l < int(ans.size())) { ans[l] = num[0]; }
      } else {
        int m = (l + r) / 2;
        work(work, 2 * id, l, m, num.mulT(q[2 * id + 1]).modxk(m - l));
        work(work, 2 * id + 1, m, r, num.mulT(q[2 * id]).modxk(r - m));
      }
    };
    work(work, 1, 0, n, mulT(q[1].inv(n)));
    return ans;
  }
};
template <int P>
Poly<P> interpolate(vector<Modint<P>> x, vector<Modint<P>> y) {
  // f(xi) = yi
  int n = x.size();
  vector<Poly<P>> p(4 * n), q(4 * n);
  auto dfs1 = [&](auto dfs1, int id, int l, int r) -> void {
    if (l == r) {
      p[id] = Poly<P>({-x[l].v, 1});
      return;
    }
    int m = l + r >> 1;
    dfs1(dfs1, id << 1, l, m);
    dfs1(dfs1, id << 1 | 1, m + 1, r);
    p[id] = p[id << 1] * p[id << 1 | 1];
  };
  dfs1(dfs1, 1, 0, n - 1);
  Poly<P> f = Poly<P>(p[1].derivative().evaluate(x));
  auto dfs2 = [&](auto dfs2, int id, int l, int r) -> void {
    if (l == r) {
      q[id] = Poly<P>({y[l] * f[l].inv()});
      return;
    }
    int m = l + r >> 1;
    dfs2(dfs2, id << 1, l, m);
    dfs2(dfs2, id << 1 | 1, m + 1, r);
    q[id] = q[id << 1] * p[id << 1 | 1] + q[id << 1 | 1] * p[id << 1];
  };
  dfs2(dfs2, 1, 0, n - 1);
  return q[1];
}
auto shift = [&](FPS f, int k) {
  FPS a(n + 1), b(n + 1);
  Mint powk = 1;
  for (int i = 0; i <= n; i++) {
    a[i] = ifact[i] * powk;
    b[i] = fact[i] * f[i];
    powk = powk * k;
  }
  reverse(b.begin(), b.end());
  auto g = a * b;
  g.resize(n + 1);
  reverse(g.begin(), g.end());
  for (int i = 0; i <= n; i++) {
    g[i] = g[i] * ifact[i];
  }
  return g;
};
```

6.4 Any Mod NTT

```cpp
constexpr int P0 = 998244353, P1 = 1004535809, P2 = 469762049;
constexpr i64 P01 = 1LL * P0 * P1;
constexpr int inv0 = Modint<P1>(P0).inv().v;
constexpr int inv01 = Modint<P2>(P01).inv().v;
for (int i = 0; i < int(c.size()); i++) {
  i64 x = 1LL * (c1[i] - c0[i] + P1) % P1 * inv0 % P1 * P0 + c0[i];
  c[i] = ((c2[i] - x % P2 + P2) % P2 * inv01 % P2 * (P01 % P) % P + x) % P;
}
```

6.5 Newton's Method

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

6.6 Fast Walsh-Hadamard Transform

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
   - $f(A) = (f(A_0) + f(A_1), f(A_1))$
   - $f^{-1}(A) = (f^{-1}(A_0) - f^{-1}(A_1), f^{-1}(A_1))$

6.7 Simplex Algorithm

Description: maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$. Returns $-\infty$ if infeasible and ∞ if unbounded.

```
const double eps = 1e-9;
const double inf = 1e+9;
int n, m;
vector<vector<double>> d;
vector<int> p, q;
void pivot(int r, int s) {
  double inv = 1.0 / d[r][s];
  for (int i = 0; i < m + 2; ++i) {
    for (int j = 0; j < n + 2; ++j) {
      if (i != r && j != s) d[i][j] -= d[r][j] * d[i][s] * inv;
    }
  }
  for (int i = 0; i < m + 2; ++i) if (i != r) d[i][s] *= -inv;
  for (int j = 0; j < n + 2; ++j) if (j != s) d[r][j] *= +inv;
  d[r][s] = inv;
  swap(p[r], q[s]);
}
bool phase(int z) {
  int x = m + z;
  while (true) {
    int s = -1;
    for (int i = 0; i <= n; ++i) {
      if (!z && q[i] == -1) continue;
      if (s == -1 || d[x][i] < d[x][s]) s = i;
    }
    if (d[x][s] > -eps) return true;
    int r = -1;
    for (int i = 0; i < m; ++i) {
      if (d[i][s] < eps) continue;
      if (r == -1 || d[i][n + 1] / d[i][s] < d[r][n + 1] / d[r][s]) r = i;
    }
    if (r == -1) return false;
    pivot(r, s);
  }
}
vector<double> solve(const vector<vector<double>> &a, const vector<double> &b, const vector<double> &c) {
  m = b.size(), n = c.size();
  d = vector<vector<double>>(m + 2, vector<double>(n + 2));
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) d[i][j] = a[i][j];
  }
  p.resize(m), q.resize(n + 1);
  for (int i = 0; i < m; ++i) p[i] = n + i, d[i][n] = -1, d[i][n + 1] = b[i];
  for (int i = 0; i < n; ++i) q[i] = i, d[m][i] = -c[i];
  q[n] = -1, d[m + 1][n] = 1;
  int r = 0;
  for (int i = 1; i < m; ++i) if (d[i][n + 1] < d[r][n + 1]) r = i;
  if (d[r][n + 1] < -eps) {
    pivot(r, n);
    if (!phase(1) || d[m + 1][n + 1] < -eps) return vector<double>(n, -inf);
    for (int i = 0; i < m; ++i) if (p[i] == -1) {
      int s = min_element(d[i].begin(), d[i].end() - 1) - d[i].begin();
      pivot(i, s);
    }
  }
  if (!phase(0)) return vector<double>(n, inf);
  vector<double> x(n);
  for (int i = 0; i < m; ++i) if (p[i] < n) x[p[i]] = d[i][n + 1];
  return x;
}
```

6.7.1 Construction

Standard form: maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge 0$.
Dual LP: minimize $\mathbf{b}^T\mathbf{y}$ subject to $A^T\mathbf{y} \ge \mathbf{c}$ and $\mathbf{y} \ge 0$.
$\bar{\mathbf{x}}$ 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. $\sum_{1\le i\le n} A_{ji}x_i \ge b_j \to \sum_{1\le i\le n} -A_{ji}x_i \le -b_j$
3. $\sum_{1\le i\le n} A_{ji}x_i = b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \le b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \ge b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.8 Subset Convolution

Description: $h(s) = \sum_{s' \subseteq s} f(s')g(s \setminus s')$

```
vector<int> SubsetConv(int n, const vector<int> &f, const vector<int> &g) {
  const int m = 1 << n;
  vector<vector<int>> a(n + 1, vector<int>(m)), b(n + 1, vector<int>(m));
  for (int i = 0; i < m; ++i) {
    a[__builtin_popcount(i)][i] = f[i];
    b[__builtin_popcount(i)][i] = g[i];
  }
  for (int i = 0; i <= n; ++i) {
    for (int j = 0; j < n; ++j) {
      for (int s = 0; s < m; ++s) {
        if (s >> j & 1) {
          a[i][s] += a[i][s ^ (1 << j)];
          b[i][s] += b[i][s ^ (1 << j)];
        }
      }
    }
  }
  vector<vector<int>> c(n + 1, vector<int>(m));
  for (int s = 0; s < m; ++s) {
    for (int i = 0; i <= n; ++i) {
      for (int j = 0; j <= i; ++j) c[i][s] += a[j][s] * b[i - j][s];
    }
  }
  for (int i = 0; i <= n; ++i) {
    for (int j = 0; j < n; ++j) {
      for (int s = 0; s < m; ++s) {
        if (s >> j & 1) c[i][s] -= c[i][s ^ (1 << j)];
      }
    }
  }
  vector<int> res(m);
  for (int i = 0; i < m; ++i) res[i] = c[__builtin_popcount(i)][i];
  return res;
}
```

6.9 Berlekamp Massey Algorithm

```
// find \sum a_(i- j)c_j = 0 for d <= i
template <typename T>
vector<T> berlekampMassey(const vector<T> &a) {
  vector<T> c(1, 1), oldC(1);
  int oldI = -1;
  T oldD = 1;
  for (int i = 0; i < int(a.size()); i++) {
    T d = 0;
    for (int j = 0; j < int(c.size()); j++) { d += c[j] * a[i - j]; }
    if (d == 0) { continue; }
    T mul = d / oldD;
    vector<T> nc = c;
    nc.resize(max(int(c.size()), i - oldI + int(oldC.size())));
    for (int j = 0; j < int(oldC.size()); j++) { nc[j + i - oldI] -= oldC[j] * mul; }
    if (i - int(c.size()) > oldI - int(oldC.size())) {
      oldI = i;
      oldD = d;
      swap(oldC, c);
    }
    swap(c, nc);
  }
  return c;
}
```

6.10 Fast Linear Recurrence

```
// p : a[0] ~ a[d - 1]
// q : a[i] = \sum a[i - j]q[j]
template <typename T>
T linearRecurrence(vector<T> p, vector<T> q, i64 n) {
  int d = q.size() - 1;
  assert(int(p.size()) == d);
  p = p * q;
  p.resize(d);
  while (n > 0) {
    auto nq = q;
    for (int i = 1; i <= d; i += 2) {
      nq[i] *= -1;
    }
    auto np = p * nq;
    nq = q * nq;
    for (int i = 0; i < d; i++) {
      p[i] = np[i * 2 + n % 2];
    }
    for (int i = 0; i <= d; i++) {
      q[i] = nq[i * 2];
    }
    n /= 2;
  }
  return p[0] / q[0];
}
```

6.11 Prime check and factorize

```
i64 mul(i64 a, i64 b, i64 mod) {}
i64 qpow(i64 x, i64 p, i64 mod) {}
bool isPrime(i64 n) {
  if (n == 1) { return false; }
  int r = __builtin_ctzll(n - 1);
  i64 d = n - 1 >> r;
  auto checkComposite = [&](i64 p) {
    i64 x = qpow(p, d, n);
    if (x == 1 || x == n - 1) { return false; }
    for (int i = 1; i < r; i++) {
      x = mul(x, x, n);
      if (x == n - 1) { return false; }
    }
    return true;
  };
  for (auto p : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
    if (n == p) {
      return true;
    } else if (checkComposite(p)) {
      return false;
    }
  }
  return true;
}
vector<i64> pollardRho(i64 n) {
  vector<i64> res;
  auto work = [&](auto work, i64 n) {
    if (n <= 10000) {
      for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
          res.push_back(i);
          n /= i;
        }
      }
      if (n > 1) { res.push_back(n); }
      return;
    } else if (isPrime(n)) {
      res.push_back(n);
      return;
    }
    i64 x0 = 2;
    auto f = [&](i64 x) { return (mul(x, x, n) + 1) % n; };
    while (true) {
      i64 x = x0, y = x0, d = 1, power = 1, lam = 0, v = 1;
```

```
      while (d == 1) {
        y = f(y);
        ++lam;
        v = mul(v, abs(x - y), n);
        if (lam % 127 == 0) {
          d = gcd(v, n);
          v = 1;
        }
        if (power == lam) {
          x = y;
          power *= 2;
          lam = 0;
          d = gcd(v, n);
          v = 1;
        }
      }
      if (d != n) {
        work(work, d);
        work(work, n / d);
        return;
      }
      ++x0;
    }
  };
  work(work, n);
  sort(res.begin(), res.end());
  return res;
}
```

6.12 Count Primes leq n

```
// __attribute__((target("avx2"),
    optimize("O3", "unroll-loops")))
i64 primeCount(const i64 n) {
  if (n <= 1) { return 0; }
  if (n == 2) { return 1; }
  const int v = sqrtl(n);
  int s = (v + 1) / 2;
  vector<int> smalls(s), roughs(s), skip(
      v + 1);
  vector<i64> larges(s);
  iota(smalls.begin(), smalls.end(), 0);
  for (int i = 0; i < s; i++) {
    roughs[i] = 2 * i + 1;
    larges[i] = (n / roughs[i] - 1) / 2;
  }
  const auto half = [](int n) -> int {
      return (n - 1) >> 1; };
  int pc = 0;
  for (int p = 3; p <= v; p += 2) {
    if (skip[p]) { continue; }
    int q = p * p;
    if (1LL * q * q > n) { break; }
    skip[p] = true;
    for (int i = q; i <= v; i += 2 * p)
        skip[i] = true;
    int ns = 0;
    for (int k = 0; k < s; k++) {
      int i = roughs[k];
      if (skip[i]) { continue; }
      i64 d = 1LL * i * p;
      larges[ns] = larges[k] - (d <= v ?
          larges[smalls[d / 2] - pc] :
          smalls[half(n / d)]) + pc;
      roughs[ns++] = i;
    }
    s = ns;
    for (int i = half(v), j = v / p - 1 |
        1; j >= p; j -= 2) {
      int c = smalls[j / 2] - pc;
      for (int e = j * p / 2; i >= e; i
          --) { smalls[i] -= c; }
    }
    pc++;
  }
  larges[0] += 1LL * (s + 2 * (pc - 1)) *
      (s - 1) / 2;
  for (int k = 1; k < s; k++) { larges[0]
      -= larges[k]; }
  for (int l = 1; l < s; l++) {
    i64 q = roughs[l];
    i64 M = n / q;
    int e = smalls[half(M / q)] - pc;
    if (e <= l) { break; }
    i64 t = 0;
    for (int k = l + 1; k <= e; k++) { t
        += smalls[half(M / roughs[k])];
    }
    larges[0] += t - 1LL * (e - l) * (pc
        + l - 1);
  }
  return larges[0] + 1;
}
```

6.13 Discrete Logarithm

```
// return min x >= 0 s.t. a ^ x = b mod m
    , 0 ^ 0 = 1, -1 if no solution
// (I think) if you want x > 0 (m != 1),
    remove if (b == k) return add;
int discreteLog(int a, int b, int m) {
  if (m == 1) {
    return 0;
  }
  a %= m, b %= m;
  int k = 1, add = 0, g;
  while ((g = gcd(a, m)) > 1) {
    if (b == k) {
      return add;
    } else if (b % g) {
      return -1;
    }
    b /= g, m /= g, ++add;
    k = 1LL * k * a / g % m;
  }
  if (b == k) {
    return add;
  }
  int n = sqrt(m) + 1;
  int an = 1;
  for (int i = 0; i < n; ++i) {
    an = 1LL * an * a % m;
  }
  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q < n; ++q) {
    vals[cur] = q;
    cur = 1LL * a * cur % m;
  }
  for (int p = 1, cur = k; p <= n; ++p) {
    cur = 1LL * cur * an % m;
    if (vals.count(cur)) {
      int ans = n * p - vals[cur] + add;
      return ans;
    }
  }
  return -1;
}
```

6.14 Quadratic Residue

```
// rng
int jacobi(int a, int m) {
  int s = 1;
  while (m > 1) {
    a %= m;
    if (a == 0) { return 0; }
    int r = __builtin_ctz(a);
    if (r % 2 == 1 && (m + 2 & 4) != 0) {
        s = -s; }
    a >>= r;
    if ((a & m & 2) != 0) { s = -s; }
    swap(a, m);
  }
  return s;
}
int quadraticResidue(int a, int p) {
  if (p == 2) { return a % 2; }
  int j = jacobi(a, p);
  if (j == 0 || j == -1) { return j; }
  int b, d;
  while (true) {
    b = rng() % 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 = p + 1 >> 1; e > 0; e >>=
      1) {
    if (e % 2 == 1) {
      tmp = (1LL * g0 * f0 + 1LL * d * g1
          % p * f1 % p) % p;
      g1 = (1LL * g0 * f1 + 1LL * g1 * f0
          ) % p;
      g0 = tmp;
    }
    tmp = (1LL * f0 * f0 + 1LL * d * f1 %
        p * f1 % p) % p;
    f1 = 2LL * f0 * f1 % p;
    f0 = tmp;
  }
  return g0;
}
```

6.15 Characteristic Polynomial

```
vector<vector<int>> Hessenberg(const
    vector<vector<int>> &A) {
  int N = A.size();
  vector<vector<int>> H = A;
  for (int i = 0; i < N - 2; ++i) {
    if (!H[i + 1][i]) {
      for (int j = i + 2; j < N; ++j) {
        if (H[j][i]) {
          for (int k = i; k < N; ++k)
              swap(H[i + 1][k], H[j][k])
              ;
          for (int k = 0; k < N; ++k)
              swap(H[k][i + 1], H[k][j])
              ;
          break;
        }
      }
    }
    if (!H[i + 1][i]) continue;
    int val = fpow(H[i + 1][i], kP - 2);
    for (int j = i + 2; j < N; ++j) {
      int coef = 1LL * val * H[j][i] % kP
          ;
      for (int k = i; k < N; ++k) H[j][k]
          = (H[j][k] + 1LL * H[i + 1][k
          ] * (kP - coef)) % kP;
      for (int k = 0; k < N; ++k) H[k][i
          + 1] = (H[k][i + 1] + 1LL * H[
          k][j] * coef) % kP;
    }
  }
  return H;
}
vector<int> CharacteristicPoly(const
    vector<vector<int>> &A) {
  int N = A.size();
  auto H = Hessenberg(A);
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) H[i][j] =
        kP - H[i][j];
  }
  vector<vector<int>> P(N + 1, vector<int
      >(N + 1));
  P[0][0] = 1;
  for (int i = 1; i <= N; ++i) {
    P[i][0] = 0;
    for (int j = 1; j <= i; ++j) P[i][j]
        = P[i - 1][j - 1];
    int val = 1;
    for (int j = i - 1; j >= 0; --j) {
      int coef = 1LL * val * H[j][i - 1]
          % kP;
      for (int k = 0; k <= j; ++k) P[i][k
          ] = (P[i][k] + 1LL * P[j][k] *
           coef) % kP;
      if (j) val = 1LL * val * (kP - H[j
          ][j - 1]) % kP;
    }
  }
  if (N & 1) {
    for (int i = 0; i <= N; ++i) P[N][i]
        = kP - P[N][i];
  }
  return P[N];
}
```

6.16 Linear Sieve Related

```
vector<int> minp(N + 1), primes, mobius(N
     + 1);
mobius[1] = 1;
for (int i = 2; i <= N; i++) {
  if (!minp[i]) {
    primes.push_back(i);
    minp[i] = i;
```

```
    mobius[i] = -1;
  }
  for (int p : primes) {
    if (p > N / i) {
      break;
    }
    minp[p * i] = p;
    mobius[p * i] = -mobius[i];
    if (i % p == 0) {
      mobius[p * i] = 0;
      break;
    }
  }
}
```

6.17 De Bruijn Sequence

```
int res[kN], aux[kN], a[kN], sz;
void Rec(int t, int p, int n, int k) {
  if (t > n) {
    if (n % p == 0)
      for (int i = 1; i <= p; ++i) res[sz
           ++] = aux[i];
  } else {
    aux[t] = aux[t - p];
    Rec(t + 1, p, n, k);
    for (aux[t] = aux[t - p] + 1; aux[t]
          < k; ++aux[t]) Rec(t + 1, t, n,
          k);
  }
}
int DeBruijn(int k, int n) {
  // return cyclic string of length k^n
       such that every string of length n
        using k character appears as a
       substring.
  if (k == 1) return res[0] = 0, 1;
  fill(aux, aux + k * n, 0);
  return sz = 0, Rec(1, 1, n, k), sz;
}
```

6.18 Floor Sum

```
// \sum {i = 0} {n} floor((a * i + b) / c
    )
i64 floorSum(i64 a, i64 b, i64 c, i64 n)
    {
  if (n < 0) { return 0; }
  if (n == 0) { return b / c; }
  if (a == 0) { return b / c * (n + 1); }
  i64 res = 0;
  if (a >= c) { res += a / c * n * (n +
      1) / 2, a %= c; }
  if (b >= c) { res += b / c * (n + 1), b
       %= c; }
  i64 m = (a * n + b) / c;
  return res + n * m - (m == 0 ? 0 :
       floorSum(c, c - b - 1, a, m - 1));
}
```

6.19 More Floor Sum

- $m = \lfloor \frac{an+b}{c} \rfloor$

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

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

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

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

6.20 Min Mod Linear

```
// \min i : [0, n) (a * i + b) % m
// ok in 1e9
int minModLinear(int n, int m, int a, int
      b, int cnt = 1, int p = 1, int q =
     1) {
  if (a == 0) { return b; }
  if (cnt % 2 == 1) {
    if (b >= a) {
      int t = (m - b + a - 1) / a;
      int c = (t - 1) * p + q;
      if (n <= c) { return b; }
      n -= c;
      b += a * t - m;
    }
    b = a - 1 - b;
  } else {
    if (b < m - a) {
      int t = (m - b - 1) / a;
      int c = t * p;
      if (n <= c) { return (n - 1) / p *
          a + b; }
      n -= c;
      b += a * t;
    }
    b = m - 1 - b;
  }
  cnt++;
  int d = m / a;
  int c = minModLinear(n, a, m % a, b,
       cnt, (d - 1) * p + q, d * p + q);
  return cnt % 2 == 1 ? m - 1 - c : a - 1
       - c;
}
```

6.21 Count of subsets with sum (mod P) leq T

```
int n, T;
cin >> n >> T;
vector<int> cnt(T + 1);
for (int i = 0; i < n; i++) {
  int a;
  cin >> a;
  cnt[a]++;
}
vector<Mint> inv(T + 1);
for (int i = 1; i <= T; i++) {
  inv[i] = i == 1 ? 1 : -P / i * inv[P %
      i];
}
FPS f(T + 1);
for (int i = 1; i <= T; i++) {
  for (int j = 1; j * i <= T; j++) {
    f[i * j] = f[i * j] + (j % 2 == 1 ? 1
        : -1) * cnt[i] * inv[j];
  }
}
f = f.exp(T + 1);
```

6.22 Theorem

- 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 non-negative integers $d_1 \ge d_2 \ge \ldots \ge d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \ldots + d_n$ is even and

  $$\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

  holds for all $1 \le k \le n$.

- Burnside's Lemma

  Let X be a set and G be a group that acts on X. For $g \in G$, denote by X^g the elements fixed by g:

  $$X^g = \{x \in X \mid gx \in X\}$$

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

- Gale–Ryser theorem

  A pair of sequences of nonnegative integers $a_1 \ge \cdots \ge a_n$ and $b_1, \ldots, b_n$ is bigraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \le \sum_{i=1}^{n} \min(b_i, k)$ holds for every $1 \le k \le n$.
  Sequences a and b called bigraphic if there is a labeled simple bipartite graph such that a and b is the degree sequence of this bipartite graph.

- Fulkerson–Chen–Anstee theorem

  A sequence $(a_1, b_1), \ldots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \ge \cdots \ge a_n$ is digraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ and $\sum_{i=1}^{k} a_i \le \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k)$ holds for every $1 \le k \le n$.
  Sequences a and b called digraphic if there is a labeled simple directed graph such that each vertex v_i has indegree a_i and outdegree b_i.

- Pick's theorem

  For simple polygon, when points are all integer, we have $A = \#\{\text{lattice points in the interior}\} + \frac{\#\{\text{lattice points on the boundary}\}}{2} - 1$

- Möbius inversion formula

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

- Spherical cap

  - A portion of a sphere cut off by a plane.
  - r: sphere radius, a: radius of the base of the cap, h: height of the cap, θ: $\arcsin(a/r)$.
  - Volume $= \pi h^2(3r - h)/3 = \pi h(3a^2 + h^2)/6 = \pi r^3(2 + \cos\theta)(1 - \cos\theta)^2/3$.
  - Area $= 2\pi rh = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos\theta)$.

- The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200000 for $n < 1e19$.
- The number of ways of writing n as a sum of positive integers, disregarding the order of the summands. $1, 1, 2, 3, 5, 7, 11, 15, 22, 30$ for $n = 0 \sim 9$, 627 for $n = 20$, $\sim 2e5$ for $n = 50$, $\sim 2e8$ for $n = 100$.

- Total number of partitions of n distinct elements: $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975,$ $27644437, 190899322, \dots$.
- Ordinary Generating Function $A(x) = \sum_{i \ge 0} a_i x^i$
  - $A(rx) \Rightarrow r^n a_n$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} a_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$
  - $xA(x)' \Rightarrow n a_n$
  - $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^{n} a_i$
- Exponential Generating Function $A(x) = \sum_{i \ge 0} \frac{a_i}{i!} x_i$
  - $A(x) + B(x) \Rightarrow a_n + b_n$
  - $A^{(k)}(x) \Rightarrow a_{n+k}$
  - $A(x)B(x) \Rightarrow \sum_{i=0}^{n} \binom{n}{i} a_i b_{n-i}$
  - $A(x)^k \Rightarrow \sum_{i_1+i_2+\cdots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k}$
  - $xA(x) \Rightarrow n a_n$
- Special Generating Function
  - $(1+x)^n = \sum_{i \ge 0} \binom{n}{i} x^i$
  - $\frac{1}{(1-x)^n} = \sum_{i \ge 0} \binom{i}{n-1} x^i$
- Bernoulli numbers

  $B_0 = 1, B_1^{\pm} = \pm\frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$

  $\sum_{j=0}^{m} \binom{m+1}{j} B_j = 0$, EGF is $B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}$.

  $S_m(n) = \sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k^{+} n^{m+1-k}$
- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.

  $S(n,k) = S(n-1,k-1) + kS(n-1,k), S(n,1) = S(n,n) = 1$

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

  $x^n = \sum_{i=0}^{n} S(n,i)(x)_i$
- Pentagonal number theorem

  $\prod_{n=1}^{\infty}(1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2}\right)$
- 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. $\pi(j) > j$.

  $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$

  $E(n,0) = E(n,n-1) = 1$

  $E(n,k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$

7 Dynamic Programming

7.1 Dynamic Convex Hull

```
struct Line {
  // kx + b
  mutable i64 k, b, p;
  bool operator<(const Line& o) const {
      return k < o.k; }
  bool operator<(i64 x) const { return p
      < x; }
};
struct DynamicConvexHullMax : multiset<
    Line, less<>> {
  // (for doubles, use INF = 1/.0, div(a,
      b) = a/b)
  static constexpr i64 INF =
      numeric_limits<i64>::max();
  i64 div(i64 a, i64 b) {
    // floor
    return a / b - ((a ^ b) < 0 && a % b)
        ;
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = INF, 0;
    if (x->k == y->k) x->p = x->b > y->b
        ? INF : -INF;
    else x->p = div(y->b - x->b, x->k - y
        ->k);
    return x->p >= y->p;
  }
  void add(i64 k, i64 b) {
    auto z = insert({k, 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));
  }
  i64 query(i64 x) {
    if (empty()) {
      return -INF;
    }
    auto l = *lower_bound(x);
    return l.k * x + l.b;
  }
};
```

7.2 $1D/1D$ Convex Optimization

```
struct segment {
  int i, l, r;
  segment(int a, int b, int c): i(a), l(b
      ), r(c) {}
};
inline long long f(int l, int r) { return
     dp[l] + w(l + 1, r); }
void solve() {
  dp[0] = 0ll;
  deque&lt;segment&gt; deq; deq.push_back(
      segment(0, 1, n));
  for (int i = 1; i <= n; ++i) {
  dp[i] = f(deq.front().i, i);
  while (deq.size() && deq.front().r < i
       + 1) deq.pop_front();
  deq.front().l = i + 1;
  segment seg = segment(i, i + 1, n);
  while (deq.size() && f(i, deq.back().l)
       < f(deq.back().i, deq.back().l))
      deq.pop_back();
  if (deq.size()) {
    int d = 1048576, c = deq.back().l;
    while (d >>= 1) if (c + d <= deq.back
        ().r) {
      if (f(i, c + d) > f(deq.back().i, c +
          d)) c += d;
    }
    deq.back().r = c; seg.l = c + 1;
  }
  if (seg.l <= n) deq.push_back(seg);
  }
}
```

7.3 Conditon

7.3.1 Totally Monotone (Concave/Convex)

$\forall i < i', j < j', B[i][j] \le B[i'][j] \implies B[i][j'] \le B[i'][j']$
$\forall i < i', j < j', B[i][j] \ge B[i'][j] \implies B[i][j'] \ge B[i'][j']$

7.3.2 Monge Condition (Concave/Convex)

$\forall i < i', j < j', B[i][j] + B[i'][j'] \ge B[i][j'] + B[i'][j]$
$\forall i < i', j < j', B[i][j] + B[i'][j'] \le B[i][j'] + B[i'][j]$

7.3.3 Optimal Split Point

If

$$B[i][j] + B[i+1][j+1] \ge B[i][j+1] + B[i+1][j]$$

then

$$H_{i,j-1} \le H_{i,j} \le H_{i+1,j}$$

8 Ckisemetry

8.1 Basic

```
#define IM imag
#define RE real
using lld = int64_t;
using llf = long double;
using PT = complex<lld>;
using PF = complex<llf>;
using P = PT;
llf abs(P p) { return sqrtl(norm(p)); }
PF toPF(PT p) { return PF{RE(p), IM(p)};
    }
int sgn(lld x) { return (x > 0) - (x < 0)
    ; }
lld dot(P a, P b) { return RE(conj(a) * b
    ); }
lld cross(P a, P b) { return IM(conj(a) *
     b); }
int ori(P a, P b, P c) {
  return sgn(cross(b - a, c - a));
}
int quad(P p) {
  return (IM(p) == 0) // use sgn for PF
    ? (RE(p) < 0 ? 3 : 1) : (IM(p) < 0 ?
      0 : 2);
}
int argCmp(P a, P b) {
  // returns 0/+-1, starts from theta = -
      PI
  int qa = quad(a), qb = quad(b);
  if (qa != qb) return sgn(qa - qb);
  return sgn(cross(b, a));
}
P rot90(P p) { return P{-IM(p), RE(p)}; }
template <typename V> llf area(const V &
    pt) {
  lld ret = 0; // BE CAREFUL OF TYPE!
  for (int i = 1; i + 1 < (int)pt.size();
       i++)
    ret += cross(pt[i] - pt[0], pt[i+1] -
         pt[0]);
  return ret / 2.0;
}
template <typename V> PF center(const V &
    pt) {
  P ret = 0; lld A = 0; // BE CAREFUL OF
      TYPE!
  for (int i = 1; i + 1 < (int)pt.size();
       i++) {
    lld cur = cross(pt[i] - pt[0], pt[i
        +1] - pt[0]);
    ret += (pt[i] + pt[i + 1] + pt[0]) *
        cur; A += cur;
  }
  return toPF(ret) / llf(A * 3);
}
PF project(PF p, PF q) { // p onto q
  return dot(p, q) * q / dot(q, q); //
      dot<llf>
}
```

8.2 ConvexHull

```
// from NaCl, counterclockwise, be
    careful of n<=2
vector<P> convex_hull(vector<P> v) { // n
    ==0 will RE
  sort(all(v)); // by X then Y
  if (v[0] == v.back()) return {v[0]};
  int t = 0, s = 1; vector<P> h(v.size()
      + 1);
  for (int _ = 2; _--; s = t--, reverse(
      all(v)))
    for (P p : v) {
      while (t>s && ori(p, h[t-1], h[t
          -2]) >= 0) t--;
      h[t++] = p;
    }
  return h.resize(t), h;
}
```

8.3 CyclicTS

```
int cyclic_ternary_search(int N, auto &&
    lt_) {
  auto lt = [&](int x, int y) {
    return lt_(x % N, y % N); };
  int l = 0, r = N; bool up = lt(0, 1);
  while (r - l > 1) {
    int m = (l + r) / 2;
    if (lt(m, 0) ? up : !lt(m, m+1)) r =
        m;
    else l = m;
  }
  return (lt(l, r) ? r : l) % N;
} // find maximum; be careful if N == 0
```

8.4 Delaunay

```
/* please ensure input points are unique
    */
/* A triangulation such that no points
    will strictly
inside circumcircle of any triangle. C
    should be big
enough s.t. the initial triangle contains
    all points */
#define L(i) ((i)==0 ? 2 : (i)-1)
#define R(i) ((i)==2 ? 0 : (i)+1)
#define F3 for (int i = 0; i < 3; i++)
bool is_inf(P z) { return RE(z) <= -C ||
    RE(z) >= C; }
bool in_cc(const array<P,3> &p, P q) {
  i128 inf_det = 0, det = 0, inf_N, N;
  F3 {
    if (is_inf(p[i]) && is_inf(q))
      continue;
    else if (is_inf(p[i])) inf_N = 1, N =
        -norm(q);
    else if (is_inf(q)) inf_N = -1, N =
        norm(p[i]);
    else inf_N = 0, N = norm(p[i]) - norm
        (q);
    lld D = cross(p[R(i)] - q, p[L(i)] -
        q);
    inf_det += inf_N * D; det += N * D;
  }
  return inf_det != 0 ? inf_det > 0 : det
      > 0;
}
P v[maxn];
struct Tri;
struct E {
  Tri *t; int side;
  E(Tri *t_=0, int side_=0) : t(t_), side
      (side_) {}
};
struct Tri {
  array<int,3> p; array<Tri*,3> ch; array
      <E,3> e;
  Tri(int a=0, int b=0, int c=0) : p{a, b
      , c}, ch{} {}
  bool has_chd() const { return ch[0] !=
      nullptr; }
  bool contains(int q) const {
    F3 if (ori(v[p[i]], v[p[R(i)]], v[q])
        < 0)
      return false;
    return true;
  }
  bool check(int q) const {
    return in_cc({v[p[0]], v[p[1]], v[p
        [2]]}, v[q]); }
} pool[maxn * 10], *it, *root;
/* SPLIT_HASH_HERE */
void link(const E &a, const E &b) {
  if (a.t) a.t->e[a.side] = b;
  if (b.t) b.t->e[b.side] = a;
}
void flip(Tri *A, int a) {
  auto [B, b] = A->e[a]; /* flip edge
      between A,B */
  if (!B || !A->check(B->p[b])) return;
  Tri *X = new (it++) Tri(A->p[R(a)], B->
      p[b], A->p[a]);
  Tri *Y = new (it++) Tri(B->p[R(b)], A->
      p[a], B->p[b]);
  link(E(X, 0), E(Y, 0));
  link(E(X, 1), A->e[L(a)]); link(E(X, 2)
      , B->e[R(b)]);
  link(E(Y, 1), B->e[L(b)]); link(E(Y, 2)
      , A->e[R(a)]);
  A->ch = B->ch = {X, Y, nullptr};
  flip(X, 1); flip(X, 2); flip(Y, 1);
      flip(Y, 2);
}
void add_point(int p) {
  Tri *r = root;
  while (r->has_chd()) for (Tri *c: r->ch
      )
    if (c && c->contains(p)) { r = c;
        break; }
  array<Tri*, 3> t; /* split into 3
      triangles */
  F3 t[i] = new (it++) Tri(r->p[i], r->p[
      R(i)], p);
  F3 link(E(t[i], 0), E(t[R(i)], 1));
  F3 link(E(t[i], 2), r->e[L(i)]);
  r->ch = t;
  F3 flip(t[i], 2);
}
auto build(const vector<P> &p) {
  it = pool; int n = (int)p.size();
  vector<int> ord(n); iota(all(ord), 0);
  shuffle(all(ord), mt19937(114514));
  root = new (it++) Tri(n, n + 1, n + 2);
  copy_n(p.data(), n, v); v[n++] = P(-C,
      -C);
  v[n++] = P(C * 2, -C); v[n++] = P(-C, C
      * 2);
  for (int i : ord) add_point(i);
  vector<array<int, 3>> res;
  for (Tri *now = pool; now != it; now++)
    if (!now->has_chd()) res.push_back(
        now->p);
  return res;
}
```

8.5 DirInPoly

```
bool DIP(const auto &p, int i, P dir) {
  const int n = (int)p.size();
  P A = p[i+1==n ? 0 : i+1] - p[i];
  P B = p[i==0 ? n-1 : i-1] - p[i];
  if (auto C = cross(A, B); C < 0)
    return cross(A, dir) >= 0 || cross(
        dir, B) >= 0;
  else
    return cross(A, dir) >= 0 && cross(
        dir, B) >= 0;
} // is Seg(p[i], p[i]+dir*eps) in p? (
    non-strict)
// p is counterclockwise simple polygon
```

8.6 FarthestPair

```
// p is CCW convex hull w/o colinear
    points
int n = (int)p.size(), pos = 1; lld ans =
    0;
for (int i = 0; i < n; i++) {
  P e = p[(i + 1) % n] - p[i];
  while (cross(e, p[(pos + 1) % n] - p[i
      ]) >
        cross(e, p[pos] - p[i]))
    pos = (pos + 1) % n;
  for (int j: {i, (i + 1) % n})
    ans = max(ans, norm(p[pos] - p[j]));
} // tested @ AOJ CGL_4_B
```

8.7 HPIGeneralLine

```
struct Line {
  lld a, b, c; // ax + by + c <= 0
  P dir() const { return P(a, b); }
  Line(lld ta, lld tb, lld tc) : a(ta), b
      (tb), c(tc) {}
  Line(P S, P T):a(IM(T-S)),b(-RE(T-S)),c
      (cross(T,S)) {}
}; using LN = const Line &;
PF intersect(LN A, LN B) {
  llf c = cross(A.dir(), B.dir());
  i128 a = i128(A.c) * B.a - i128(B.c) *
      A.a;
  i128 b = i128(A.c) * B.b - i128(B.c) *
      A.b;
  return PF(-b / c, a / c);
}
bool cov(LN l, LN A, LN B) {
  i128 c = cross(A.dir(), B.dir());
  i128 a = i128(A.c) * B.a - i128(B.c) *
      A.a;
  i128 b = i128(A.c) * B.b - i128(B.c) *
      A.b;
  return sgn(a * l.b - b * l.a + c * l.c)
      * sgn(c) >= 0;
}
bool operator<(LN a, LN b) {
  if (int c = argCmp(a.dir(), b.dir()))
    return c == -1;
  return i128(abs(b.a) + abs(b.b)) * a.c
      >
                i128(abs(a.a) + abs(a.b
      )) * b.c;
}
```

8.8 HalfPlaneIntersection

```
struct Line {
  P st, ed, dir;
  Line (P s, P e) : st(s), ed(e), dir(e -
      s) {}
}; using LN = const Line &;
PF intersect(LN A, LN B) {
  llf t = cross(B.st - A.st, B.dir) /
    llf(cross(A.dir, B.dir));
  return toPF(A.st) + toPF(A.dir) * t; //
      C^3 / C^2
}
bool cov(LN l, LN A, LN B) {
  i128 u = cross(B.st-A.st, B.dir);
  i128 v = cross(A.dir, B.dir);
  // ori(l.st, l.ed, A.st + A.dir*(u/v))
      <= 0?
  i128 x = RE(A.dir) * u + RE(A.st - l.st
      ) * v;
  i128 y = IM(A.dir) * u + IM(A.st - l.st
      ) * v;
  return sgn(x*IM(l.dir) - y*RE(l.dir)) *
      sgn(v) >= 0;
} // x, y are C^3, also sgn<i128> is
    needed
bool operator<(LN a, LN b) {
  if (int c = argCmp(a.dir, b.dir))
    return c == -1;
  return ori(a.st, a.ed, b.st) < 0;
}
// cross(pt-line.st, line.dir)<=0 <-> pt
    in half plane
// the half plane is the LHS when going
    from st to ed
llf HPI(vector<Line> &q) {
  sort(q.begin(), q.end());
  int n = (int)q.size(), l = 0, r = -1;
  for (int i = 0; i < n; i++) {
    if (i && !argCmp(q[i].dir, q[i-1].dir
        )) continue;
    while (l < r && cov(q[i], q[r-1], q[r
        ])) --r;
    while (l < r && cov(q[i], q[l], q[l
        +1])) ++l;
    q[++r] = q[i];
  }
  while (l < r && cov(q[l], q[r-1], q[r])
      ) --r;
  while (l < r && cov(q[r], q[l], q[l+1])
      ) ++l;
  n = r - l + 1; // q[l .. r] are the
      lines
  if (n <= 2 || !argCmp(q[l].dir, q[r].
      dir)) return 0;
  vector<PF> pt(n);
  for (int i = 0; i < n; i++)
    pt[i] = intersect(q[i+l], q[(i+1)%n+l
        ]);
  return area(pt);
} // test @ 2020 Nordic NCPC : BigBrother
```

8.9 HullCut

```
vector<P> cut(const vector<P> &p, P s, P e) {
  vector<P> res;
  for (size_t i = 0; i < p.size(); i++) {
    P cur = p[i], prv = i ? p[i-1] : p.back();
    bool side = ori(s, e, cur) > 0;
    if (side != (ori(s, e, prv) > 0))
      res.push_back(intersect({s, e}, {cur, prv}));
    if (side) res.push_back(cur);
  } // P is complex<llf>
  return res; // hull intersection with halfplane
} // left of the line s -> e
```

8.10 KDTree

```
struct KDTree {
  struct Node {
    int x, y, x1, y1, x2, y2, id, f; Node *L, *R;
  } tree[maxn], *root;
  lld dis2(int x1, int y1, int x2, int y2) {
    lld dx = x1 - x2, dy = y1 - y2;
    return dx * dx + dy * dy;
  }
  static bool cmpx(Node& a, Node& b) { return a.x<b.x; }
  static bool cmpy(Node& a, Node& b) { return a.y<b.y; }
  void init(vector<pair<int,int>> &ip) {
    for (int i = 0; i < ssize(ip); i++)
      tie(tree[i].x, tree[i].y) = ip[i], tree[i].id = i;
    root = build(0, (int)ip.size()-1, 0);
  }
  Node* build(int L, int R, int d) {
    if (L>R) return nullptr;
    int M = (L+R)/2;
    nth_element(tree+L,tree+M,tree+R+1,d%2?cmpy:cmpx);
    Node &o = tree[M]; o.f = d % 2;
    o.x1 = o.x2 = o.x; o.y1 = o.y2 = o.y;
    o.L = build(L, M-1, d+1); o.R = build(M+1, R, d+1);
    for (Node *s: {o.L, o.R}) if (s) {
      o.x1 = min(o.x1, s->x1); o.x2 = max(o.x2, s->x2);
      o.y1 = min(o.y1, s->y1); o.y2 = max(o.y2, s->y2);
    }
    return tree+M;
  }
  bool touch(int x, int y, lld d2, Node *r){
    lld d = (lld)sqrt(d2)+1;
    return x >= r->x1 - d && x <= r->x2 + d &&
           y >= r->y1 - d && y <= r->y2 + d;
  }
  using P = pair<lld, int>;
  void dfs(int x, int y, P &mn, Node *r) {
    if (!r || !touch(x, y, mn.first, r)) return;
    mn = min(mn, P(dis2(r->x, r->y, x, y), r->id));
    if (r->f == 1 ? y < r->y : x < r->x)
      dfs(x, y, mn, r->L), dfs(x, y, mn, r->R);
    else
      dfs(x, y, mn, r->R), dfs(x, y, mn, r->L);
  }
  int query(int x, int y) {
    P mn(INF, -1); dfs(x, y, mn, root);
    return mn.second;
  }
} tree;
```

8.11 MinMaxEnclosingRect

```
// from 8BQube, plz ensure p is strict convex hull
const llf INF = 1e18, qi = acos(-1) / 2 * 3;
pair<llf, llf> solve(const vector<P> &p) {
  llf mx = 0, mn = INF; int n = (int)p.size();
  for (int i = 0, u = 1, r = 1, l = 1; i < n; ++i) {
#define Z(v) (p[(v) % n] - p[i])
    P e = Z(i + 1);
    while (cross(e, Z(u + 1)) > cross(e, Z(u))) ++u;
    while (dot(e, Z(r + 1)) > dot(e, Z(r))) ++r;
    if (!i) l = r + 1;
    while (dot(e, Z(l + 1)) < dot(e, Z(l))) ++l;
    P D = p[r % n] - p[l % n];
    llf H = cross(e, Z(u)) / llf(norm(e));
    mn = min(mn, dot(e, D) * H);
    llf B = sqrt(norm(D)) * sqrt(norm(Z(u)));
    llf deg = (qi - acos(dot(D, Z(u)) / B)) / 2;
    mx = max(mx, B * sin(deg) * sin(deg));
  }
  return {mn, mx};
} // test @ UVA 819
```

8.12 MinkowskiSum

```
// A, B are strict convex hull rotate to min by (X, Y)
vector<P> Minkowski(vector<P> A, vector<P> B) {
  const int N = (int)A.size(), M = (int)B.size();
  vector<P> sa(N), sb(M), C(N + M + 1);
  for (int i = 0; i < N; i++) sa[i] = A[(i+1)%N]-A[i];
  for (int i = 0; i < M; i++) sb[i] = B[(i+1)%M]-B[i];
  C[0] = A[0] + B[0];
  for (int i = 0, j = 0; i < N || j < M; ) {
    P e = (j>=M || (i<N && cross(sa[i], sb[j])>=0))
      ? sa[i++] : sb[j++];
    C[i + j] = e;
  }
  partial_sum(all(C), C.begin()); C.pop_back();
  return convex_hull(C); // just to remove colinear
} // be careful if min(|A|,|B|)<=2
```

8.13 PointInHull

```
bool isAnti(P a, P b) {
  return cross(a, b) == 0 && dot(a, b) <= 0; }
bool PIH(const vector<P> &h, P z, bool strict = true) {
  int n = (int)h.size(), a = 1, b = n - 1, r = !strict;
  if (n < 3) return r && isAnti(h[0] - z, h[n-1] - z);
  if (ori(h[0],h[a],h[b]) > 0) swap(a, b);
  if (ori(h[0],h[a],z) >= r || ori(h[0],h[b],z) <= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (ori(h[0], h[c], z) > 0 ? b : a) = c;
  }
  return ori(h[a], h[b], z) < r;
}
```

8.14 PointInPoly

```
bool PIP(const vector<P> &p, P z, bool strict = true) {
  int cnt = 0, n = (int)p.size();
  for (int i = 0; i < n; i++) {
    P A = p[i], B = p[(i + 1) % n];
    if (isInter(Seg(A, B), z)) return !strict;
    auto zy = IM(z), Ay = IM(A), By = IM(B);
    cnt ^= ((zy<Ay) - (zy<By)) * ori(z, A, B) > 0;
  }
  return cnt;
}
```

8.15 PointInPolyFast

```
vector<int> PIPfast(vector<P> p, vector<P> q) {
  const int N = int(p.size()), Q = int(q.size());
  vector<pair<P, int>> evt; vector<Seg> edge;
  for (int i = 0; i < N; i++) {
    int a = i, b = (i + 1) % N;
    P A = p[a], B = p[b];
    assert (A < B || B < A); // std::operator<
    if (B < A) swap(A, B);
    evt.emplace_back(A, i); evt.emplace_back(B, ~i);
    edge.emplace_back(A, B);
  }
  for (int i = 0; i < Q; i++)
    evt.emplace_back(q[i], i + N);
  sort(all(evt));
  auto vtx = p; sort(all(vtx));
  auto eval = [](const Seg &a, lld x) -> llf {
    if (RE(a.dir) == 0) {
      assert (x == RE(a.st));
      return IM(a.st) + llf(IM(a.dir)) / 2;
    }
    llf t = (x - RE(a.st)) / llf(RE(a.dir));
    return IM(a.st) + IM(a.dir) * t;
  };
  lld cur_x = 0;
  auto cmp = [&](const Seg &a, const Seg &b) -> bool {
    if (int s = sgn(eval(a, cur_x) - eval(b, cur_x)))
      return s == -1; // be careful: sgn<llf>, sgn<lld>
    int s = sgn(cross(b.dir, a.dir));
    if (cur_x != RE(a.st) && cur_x != RE(b.st)) s *= -1;
    return s == -1;
  };
  namespace pbds = __gnu_pbds;
  pbds::tree<Seg, int, decltype(cmp),
    pbds::rb_tree_tag,
    pbds::
        tree_order_statistics_node_update> st(cmp);
  auto answer = [&](P ep) {
    if (binary_search(all(vtx), ep))
      return 1; // on vertex
    Seg H(ep, ep); // ??
    auto it = st.lower_bound(H);
    if (it != st.end() && isInter(it->first, ep))
      return 1; // on edge
    if (it != st.begin() && isInter(prev(it)->first, ep))
      return 1; // on edge
    auto rk = st.order_of_key(H);
    return rk % 2 == 0 ? 0 : 2; // 0: outside, 2: inside
  };
  vector<int> ans(Q);
  for (auto [ep, i] : evt) {
    cur_x = RE(ep);
    if (i < 0) { // remove
      st.erase(edge[~i]);
    } else if (i < N) { // insert
```

```
      auto [it, succ] = st.insert({edge[i], i});
      assert(succ);
    } else ans[i - N] = answer(ep);
  }
  return ans;
} // test @ AOJ CGL_3_C
```

8.16 PolyUnion

```
llf polyUnion(const vector<vector<P>> &p) {
  vector<tuple<P, P, int>> seg;
  for (int i = 0; i < ssize(p); i++)
    for (int j = 0, m = int(p[i].size()); j < m; j++)
      seg.emplace_back(p[i][j], p[i][(j + 1) % m], i);
  llf ret = 0; // area of p[i] must be non-negative
  for (auto [A, B, i] : seg) {
    vector<pair<llf, int>> evt{{0, 0}, {1, 0}};
    for (auto [C, D, j] : seg) {
      int sc = ori(A, B, C), sd = ori(A, B, D);
      if (sc != sd && i != j && min(sc, sd) < 0) {
        llf sa = cross(D-C, A-C), sb = cross(D-C, B-C);
        evt.emplace_back(sa / (sa - sb), sgn(sc - sd));
      } else if (!sc && !sd && j < i && sgn(dot(B - A, D - C)) > 0) {
        evt.emplace_back(real((C - A) / (B - A)), 1);
        evt.emplace_back(real((D - A) / (B - A)), -1);
      }
    }
    for (auto &[q, _] : evt) q = clamp<llf>(q, 0, 1);
    sort(evt.begin(), evt.end());
    llf sum = 0, last = 0; int cnt = 0;
    for (auto [q, c] : evt) {
      if (!cnt) sum += q - last;
      cnt += c; last = q;
    }
    ret += cross(A, B) * sum;
  }
  return ret / 2;
}
```

8.17 RotatingSweepLine

```
struct Event {
  P d; int u, v;
  bool operator<(const Event &b) const {
    return sgn(cross(d, b.d)) > 0; }
};
P makePositive(P z) { return cmpxy(z, 0) ? -z : z; }
void rotatingSweepLine(const vector<P> &p) {
  const int n = int(p.size());
  vector<Event> e; e.reserve(n * (n - 1) / 2);
  for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
      e.emplace_back(makePositive(p[i] - p[j]), i, j);
  sort(all(e));
  vector<int> ord(n), pos(n);
  iota(all(ord), 0);
  sort(all(ord), [&p](int i, int j) {
    return cmpxy(p[i], p[j]); });
  for (int i = 0; i < n; i++) pos[ord[i]] = i;
  const auto makeReverse = [](auto &v) {
    sort(all(v)); v.erase(unique(all(v)), v.end());
    vector<pair<int,int>> segs;
    for (size_t i = 0, j = 0; i < v.size(); i = j) {
      for (; j < v.size() && v[j] - v[i] <= j - i; j++);
      segs.emplace_back(v[i], v[j - 1] + 1 + 1);
    }
    return segs;
  };
  for (size_t i = 0, j = 0; i < e.size(); i = j) {
    /* do here */
    vector<size_t> tmp;
    for (; j < e.size() && !(e[i] < e[j]); j++)
      tmp.push_back(min(pos[e[j].u], pos[e[j].v]));
    for (auto [l, r] : makeReverse(tmp)) {
      reverse(ord.begin() + l, ord.begin() + r);
      for (int t = l; t < r; t++) pos[ord[t]] = t;
    }
  }
}
```

8.18 SegIsIntersect

```
struct Seg { // closed segment
  P st, dir; // represent st + t*dir for 0<=t<=1
  Seg(P s, P e) : st(s), dir(e - s) {}
  static bool valid(lld p, lld q) {
    // is there t s.t. 0 <= t <= 1 && qt == p ?
    if (q < 0) q = -q, p = -p;
    return sgn(0 - p) <= 0 && sgn(p - q) <= 0;
  }
  vector<P> ends() const { return { st, st + dir }; }
};
template <typename T> bool isInter(T A, P p) {
  if (sgn(norm(A.dir)) == 0)
    return sgn(norm(p - A.st)) == 0; // BE CAREFUL
  return sgn(cross(p - A.st, A.dir)) == 0 &&
    T::valid(dot(p - A.st, A.dir), norm(A.dir));
}
template <typename U, typename V>
bool isInter(U A, V B) {
  if (sgn(cross(A.dir, B.dir)) == 0) { // BE CAREFUL
    bool res = false;
    for (P p: A.ends()) res |= isInter(B, p);
    for (P p: B.ends()) res |= isInter(A, p);
    return res;
  }
  P D = B.st - A.st; lld C = cross(A.dir, B.dir);
  return U::valid(cross(D, B.dir), C) &&
    V::valid(cross(D, A.dir), C);
}
```

8.19 SegSegDist

```
// be careful of abs<complex<int>> (replace _abs below)
llf PointSegDist(P A, Seg B) {
  if (B.dir == P(0)) return _abs(A - B.st);
  if (sgn(dot(A - B.st, B.dir)) *
      sgn(dot(A - B.ed, B.dir)) <= 0)
    return abs(cross(A - B.st, B.dir)) / _abs(B.dir);
  return min(_abs(A - B.st), _abs(A - B.ed));
}
llf SegSegDist(const Seg &s1, const Seg &s2) {
  if (isInter(s1, s2)) return 0;
  return min({
      PointSegDist(s1.st, s2),
      PointSegDist(s1.ed, s2),
      PointSegDist(s2.st, s1),
      PointSegDist(s2.ed, s1) });
} // test @ QOJ2444 / PTZ19 Summer.D3
```

8.20 SimulateAnnealing

```
llf anneal() {
  mt19937 rnd_engine(seed);
  uniform_real_distribution<llf> rnd(0, 1);
  const llf dT = 0.001;
  // Argument p
  llf S_cur = calc(p), S_best = S_cur;
  for (llf T = 2000; T > EPS; T -= dT) {
    // Modify p to p_prime
    const llf S_prime = calc(p_prime);
    const llf delta_c = S_prime - S_cur;
    llf prob = min((llf)1, exp(-delta_c / T));
    if (rnd(rnd_engine) <= prob)
      S_cur = S_prime, p = p_prime;
    if (S_prime < S_best) // find min
      S_best = S_prime, p_best = p_prime;
  }
  return S_best;
}
```

8.21 TangentPointToHull

```
pair<int, int> get_tangent(const vector<P> &v, P p) {
  auto gao = [&](int s) {
    return cyclic_ternary_search(v.size(),
      [&](int x, int y) {
        return ori(p, v[x], v[y]) == s;
      });
  }; // test @ codeforces.com/gym/101201/problem/E
  return {gao(1), gao(-1)}; // (a,b):ori(p,v[a],v[b])<0
} // plz ensure that point strictly out of hull
  // if colinear, returns arbitrary point on line
```

8.22 TriCenter

```
O = ... // see min circle cover
G = (A + B + C) / 3;
H = G * 3 - O * 2; // orthogonal center
llf a = abs(B - C), b = abs(A - C), c = abs(A - B);
I = (a * A + b * B + c * C) / (a + b + c);
// FermatPoint: minimizes sum of distance
// if max. angle >= 120 deg then vertex
// otherwise, make eq. triangle AB'C, CA'B, BC'A
// line AA', BB', CC' intersects at P
```

8.23 Voronoi

```
void build_voronoi_cells(auto &&p, auto &&res) {
  vector<vector<int>> adj(p.size());
  for (auto f: res) F3 {
    int a = f[i], b = f[R(i)];
    if (a >= p.size() || b >= p.size())
      continue;
    adj[a].emplace_back(b);
  }
  // use `adj` and `p` and HPI to build cells
  for (size_t i = 0; i < p.size(); i++) {
    vector<Line> ls = frame; // the frame
    for (int j : adj[i]) {
      P m = p[i] + p[j], d = rot90(p[j] - p[i]);
      assert (norm(d) != 0);
      ls.emplace_back(m, m + d); // doubled coordinate
    } // HPI(ls)
  }
}
```

9 Miscellaneous

9.1 Cactus 1

```
auto work = [&](const vector<int> cycle)
     {
  // merge cycle info to u?
  int len = cycle.size(), u = cycle[0];
};
auto dfs = [&](auto dfs, int u, int p) {
  par[u] = p;
  vis[u] = 1;
  for (auto v : adj[u]) {
    if (v == p) { continue; }
    if (vis[v] == 0) {
      dfs(dfs, v, u);
      if (!cyc[v]) { // merge dp }
    } else if (vis[v] == 1) {
      for (int w = u; w != v; w = par[w])
          {
        cyc[w] = 1;
      }
    } else {
      vector<int> cycle = {u};
      for (int w = v; w != u; w = par[w])
          { cycle.push_back(w); }
      work(cycle);
    }
  }

  vis[u] = 2;
};
```

9.2 Cactus 2

```
// a component contains no articulation
     point, so P2 is a component
// but not a vertex biconnected component
     by definition
// resulting bct is rooted
struct BlockCutTree {
  int n, square = 0, cur = 0;
  vector<int> low, dfn, stk;
  vector<vector<int>> adj, bct;
  BlockCutTree(int n) : n(n), low(n), dfn
       (n, -1), adj(n), bct(n) {}
  void build() { dfs(0); }
  void addEdge(int u, int v) { adj[u].
       push_back(v), adj[v].push_back(u);
       }
  void dfs(int u) {
    low[u] = dfn[u] = cur++;
    stk.push_back(u);
    for (auto v : adj[u]) {
      if (dfn[v] == -1) {
        dfs(v);
        low[u] = min(low[u], low[v]);
        if (low[v] == dfn[u]) {
          bct.emplace_back();
          int x;
          do {
            x = stk.back();
            stk.pop_back();
            bct.back().push_back(x);
          } while (x != v);
          bct[u].push_back(n + square);
          square++;
        }
      } else {
        low[u] = min(low[u], dfn[v]);
      }
    }
  }
};
```

9.3 Dancing Links

```
#include <bits/stdc++.h>
using namespace std;
// tioj 1333
#define TRAV(i, link, start) for (int i =
      link[start]; i != start; i = link[i
     ])
const int NN = 40000, RR = 200;
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;
  }
};
int main() {
    int n, m; cin >> n >> m;
    DLX<true> solver;
    solver.init(m);
    for (int i = 0; i < n; i++){
        vector<int> add;
        for (int j = 0; j < m; j++){
            int x; cin >> x;
            if (x == 1) {
                add.push_back(j);
            }
        }
        solver.insert(add);
    }
    cout << solver.solve() << '\n';
    return 0;
}
```

9.4 Offline Dynamic MST

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

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

void solve(int l, int r, vector<int> v,
     long long c) {
  if (l == r) {
    cost[qr[l].first] = qr[l].second;
    if (st[qr[l].first] == ed[qr[l].first
         ]) {
      printf("%lld\n", c);
```

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

9.5 Matroid Intersection

- $x \to y$ if $S - \{x\} \cup \{y\} \in I_1$ with $cost(\{y\})$.
- $source \to y$ if $S \cup \{y\} \in I_1$ with $cost(\{y\})$.
- $y \to x$ if $S - \{x\} \cup \{y\} \in I_2$ with $-cost(\{y\})$.
- $y \to sink$ if $S \cup \{y\} \in I_2$ with $-cost(\{y\})$.

Augmenting path is shortest path from source to sink.

9.6 Euler Tour

```
vector<int> euler, vis(V);
auto dfs = [&](auto dfs, int u) -> void {
  while (!adj[u].empty()) {
    while (!adj[u].empty() && del[adj[u].back()[1]]) {
      adj[u].pop_back();
    }
    if (!adj[u].empty()) {
      auto [v, i] = adj[u].back();
      del[i] = true;
      dfs(dfs, v);
    }
  }
  euler.push_back(u);
};
dfs(dfs, 0);
reverse(euler.begin(), euler.end());
```

9.7 SegTree Beats

```
struct SegmentTree {
  int n;
  struct node {
    i64 mx1, mx2, mxc;
    i64 mn1, mn2, mnc;
    i64 add;
    i64 sum;
    node(i64 v = 0) {
      mx1 = mn1 = sum = v;
      mxc = mnc = 1;
      add = 0;
      mx2 = -9e18, mn2 = 9e18;
    }
  };
  vector<node> t;
  // build
  void push(int id, int l, int r) {
    auto& c = t[id];
    int m = l + r >> 1;
    if (c.add != 0) {
      apply_add(id << 1, l, m, c.add);
      apply_add(id << 1 | 1, m + 1, r, c.add);
      c.add = 0;
    }
    apply_min(id << 1, l, m, c.mn1);
    apply_min(id << 1 | 1, m + 1, r, c.mn1);
    apply_max(id << 1, l, m, c.mx1);
    apply_max(id << 1 | 1, m + 1, r, c.mx1);
  }
  void apply_add(int id, int l, int r, i64 v) {
    if (v == 0) {
      return;
    }
    auto& c = t[id];
    c.add += v;
    c.sum += v * (r - l + 1);
    c.mx1 += v;
    c.mn1 += v;
    if (c.mx2 != -9e18) {
      c.mx2 += v;
    }
    if (c.mn2 != 9e18) {
      c.mn2 += v;
    }
  }
  void apply_min(int id, int l, int r, i64 v) {
    auto& c = t[id];
    if (v <= c.mn1) {
      return;
    }
    c.sum -= c.mn1 * c.mnc;
    c.mn1 = v;
    c.sum += c.mn1 * c.mnc;
    if (l == r || v >= c.mx1) {
      c.mx1 = v;
    } else if (v > c.mx2) {
      c.mx2 = v;
    }
  }
  void apply_max(int id, int l, int r, i64 v) {
    auto& c = t[id];
    if (v >= c.mx1) {
      return;
    }
    c.sum -= c.mx1 * c.mxc;
    c.mx1 = v;
    c.sum += c.mx1 * c.mxc;
    if (l == r || v <= c.mn1) {
      c.mn1 = v;
    } else if (v < c.mn2) {
      c.mn2 = v;
    }
  }
  void pull(int id) {
    auto &c = t[id], &lc = t[id << 1], &rc = t[id << 1 | 1];
    c.sum = lc.sum + rc.sum;
    if (lc.mn1 == rc.mn1) {
      c.mn1 = lc.mn1;
      c.mn2 = min(lc.mn2, rc.mn2);
      c.mnc = lc.mnc + rc.mnc;
    } else if (lc.mn1 < rc.mn1) {
      c.mn1 = lc.mn1;
      c.mn2 = min(lc.mn2, rc.mn1);
      c.mnc = lc.mnc;
    } else {
      c.mn1 = rc.mn1;
      c.mn2 = min(lc.mn1, rc.mn2);
      c.mnc = rc.mnc;
    }
    if (lc.mx1 == rc.mx1) {
      c.mx1 = lc.mx1;
      c.mx2 = max(lc.mx2, rc.mx2);
      c.mxc = lc.mxc + rc.mxc;
    } else if (lc.mx1 > rc.mx1) {
      c.mx1 = lc.mx1;
      c.mx2 = max(lc.mx2, rc.mx1);
      c.mxc = lc.mxc;
    } else {
      c.mx1 = rc.mx1;
      c.mx2 = max(lc.mx1, rc.mx2);
      c.mxc = rc.mxc;
    }
  }
  void range_chmin(int id, int l, int r, int ql, int qr, i64 v) {
    if (r < ql || l > qr || v >= t[id].mx1) {
      return;
    }
    if (ql <= l && r <= qr && v > t[id].mx2) {
      apply_max(id, l, r, v);
      return;
    }
    push(id, l, r);
    int m = l + r >> 1;
    range_chmin(id << 1, l, m, ql, qr, v);
    range_chmin(id << 1 | 1, m + 1, r, ql, qr, v);
    pull(id);
  }
  void range_chmin(int ql, int qr, i64 v) {
    range_chmin(1, 0, n - 1, ql, qr, v);
  }
  void range_chmax(int id, int l, int r, int ql, int qr, i64 v) {
    if (r < ql || l > qr || v <= t[id].mn1) {
      return;
    }
    if (ql <= l && r <= qr && v < t[id].mn2) {
      apply_min(id, l, r, v);
      return;
    }
    push(id, l, r);
    int m = l + r >> 1;
    range_chmax(id << 1, l, m, ql, qr, v);
    range_chmax(id << 1 | 1, m + 1, r, ql, qr, v);
    pull(id);
  }
  void range_chmax(int ql, int qr, i64 v) {
    range_chmax(1, 0, n - 1, ql, qr, v);
  }
  void range_add(int id, int l, int r, int ql, int qr, i64 v) {
    if (r < ql || l > qr) {
      return;
    }
    if (ql <= l && r <= qr) {
      apply_add(id, l, r, v);
      return;
    }
    push(id, l, r);
    int m = l + r >> 1;
    range_add(id << 1, l, m, ql, qr, v);
    range_add(id << 1 | 1, m + 1, r, ql, qr, v);
    pull(id);
  }
  void range_add(int ql, int qr, i64 v) {
    range_add(1, 0, n - 1, ql, qr, v);
  }
  i64 range_sum(int id, int l, int r, int ql, int qr) {
    if (r < ql || l > qr) {
      return 0;
    }
    if (ql <= l && r <= qr) {
      return t[id].sum;
    }
    push(id, l, r);
    int m = l + r >> 1;
```

```
    return range_sum(id << 1, l, m, ql,
         qr) + range_sum(id << 1 | 1, m +
          1, r, ql, qr);
  }
  i64 range_sum(int ql, int qr) {
    return range_sum(1, 0, n - 1, ql, qr)
         ;
  }
};
```

9.8 Decimal

```
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
```

9.9 AdaSimpson

```
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}; }
```

9.10 SB Tree

```
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;
}
```

9.11 Bitset LCS

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

9.12 Hilbert Curve

```
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
```

9.13 Mo on Tree

```
void MoAlgoOnTree() {
  Dfs(0, -1);
  vector<int> euler(tk);
  for (int i = 0; i < n; ++i) {
    euler[tin[i]] = i;
    euler[tout[i]] = i;
  }
  vector<int> l(q), r(q), qr(q), sp(q,
       -1);
  for (int i = 0; i < q; ++i) {
    if (tin[u[i]] > tin[v[i]]) swap(u[i],
          v[i]);
    int z = GetLCA(u[i], v[i]);
    sp[i] = z[i];
    if (z == u) l[i] = tin[u[i]], r[i] =
         tin[v[i]];
    else l[i] = tout[u[i]], r[i] = tin[v[
         i]];
    qr[i] = i;
  }
  sort(qr.begin(), qr.end(), [&](int i,
       int j) {
      if (l[i] / kB == l[j] / kB) return
           r[i] < r[j];
      return l[i] / kB < l[j] / kB;
      });
  vector<bool> used(n);
  // Add(v): add/remove v to/from the
       path based on used[v]
  for (int i = 0, tl = 0, tr = -1; i < q;
        ++i) {
    while (tl < l[qr[i]]) Add(euler[tl
         ++]);
    while (tl > l[qr[i]]) Add(euler[--tl
         ]);
    while (tr > r[qr[i]]) Add(euler[tr
         --]);
    while (tr < r[qr[i]]) Add(euler[++tr
         ]);
    // add/remove LCA(u, v) if necessary
  }
}
```

9.14 N Queens

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

9.15 Rollback Mo

```
for (int l = 0, r = -1; auto [ql, qr, i]
      : qs) {
  if (ql / B == qr / B) {
    for (int j = ql; j <= qr; j++) {
      cntSmall[a[j]]++;
      ans[i] = max(ans[i], 1LL * b[a[j]]
           * cntSmall[a[j]]);
    }
    for (int j = ql; j <= qr; j++) {
      cntSmall[a[j]]--;
    }
    continue;
  }
  if (int block = ql / B; block != lst) {
    int x = min((block + 1) * B, n);
    while (r + 1 < x) { add(++r); }
    while (r >= x) { del(r--); }
    while (l < x) { del(l++); }
    mx = 0;
    lst = block;
  }
  while (r < qr) { add(++r); }
  i64 tmpMx = mx;
  int tmpL = l;
  while (l > ql) { add(--l); }
  ans[i] = mx;
  mx = tmpMx;
  while (l < tmpL) { del(l++); }
}
```

9.16 Subset Sum

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