Contents

1 Basic

1.1 vimrc

```
se nu rnu bs=2 ru mouse=a cin et ts=2 sw=2 sts=2
syn on
colo desert
filetype indent on
inoremap {<CR> {<CR>}<Esc>O
" setxkbmap -option ctrl:nocaps
```

1.2 Fast Integer Input

```cpp
inline int gtx() {
  const int N = 4096;
  static char buffer[N];
  static char *p = buffer, *end = buffer;
  if (p == end) {
    if ((end = buffer + fread(buffer, 1, N, stdin)) == buffer)
     return EOF;
    p = buffer;
  }
  return *p++;
}
template <typename T>
inline bool rit(T& x) {
  char c = 0; bool flag = false;
  while (c = getchar(), (c < '0' && c != '-') || c > '9') if (c
      == -1) return false;
  c == '-' ? (flag = true, x = 0) : (x = c - '0');
  while (c = getchar(), c >= '0' && c <= '9') x = x * 10 + c -
     '0';
  if (flag) x = -x;
  return true;
}
```

1.3 Increase stack size

```cpp
const int size = 256 << 20;
register long rsp asm("rsp");
char *p = (char*)malloc(size) + size, *bak = (char*)rsp;
__asm__("movq %0, %%rsp\n"::"r"(p));
// main
__asm__("movq %0, %%rsp\n"::"r"(bak));
```

1.4 Pragma optimization

```cpp
#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 Dinic's Algorithm

```
struct Edge {
  int to, cap, rev;
  Edge(int t, int c, int r) : to(t), cap(c), rev(r) {}
};
int Flow(vector<vector<Edge>> g, int s, int t) {
  int n = g.size(), res = 0;
  vector<int> lev(n, -1), iter(n);
  while (true) {
    vector<int> que(1, s);
    fill(lev.begin(), lev.end(), -1);
    fill(iter.begin(), iter.end(), 0);
    lev[s] = 0;
    for (int it = 0; it < que.size(); ++it) {
      int x = que[it];
      for (Edge &e : g[x]) {
        if (e.cap > 0 && lev[e.to] == -1) {
          lev[e.to] = lev[x] + 1;
          que.push_back(e.to);
        }
      }
    }
    if (lev[t] == -1) break;
    auto Dfs = [&](auto dfs, int x, int f = 1000000000) {
      if (x == t) return f;
      int res = 0;
      for (int &it = iter[x]; it < g[x].size(); ++it) {
        Edge &e = g[x][it];
        if (e.cap > 0 && lev[e.to] == lev[x] + 1) {
          int p = dfs(dfs, e.to, min(f - res, e.cap));
          res += p;
          e.cap -= p;
          g[e.to][e.rev].cap += p;
        }
      }
      if (res == 0) lev[x] = -1;
      return res;
    };
    res += Dfs(Dfs, s);
  }
  return res;
}
```

2.2 Minimum-cost flow

```
struct Edge {
  int to, cap, rev, w;
  Edge(int t, int c, int r, int w) : to(t), cap(c), rev(r), w(w
    ) {}
};
pair<int, int> Flow(vector<vector<Edge>> g, int s, int t) {
  int N = g.size();
  vector<int> dist(N), ed(N), pv(N);
  vector<bool> inque(N);
  int flow = 0, cost = 0;
  while (true) {
    dist.assign(N, kInf);
    inque.assign(N, false);
    pv.assign(N, -1);
    dist[s] = 0;
    queue<int> que;
    que.push(s);
    while (!que.empty()) {
      int x = que.front(); que.pop();
      inque[x] = false;
      for (int i = 0; i < g[x].size(); ++i) {
        Edge &e = g[x][i];
        if (e.cap > 0 && dist[e.to] > dist[x] + e.w) {
          dist[e.to] = dist[x] + e.w;
          pv[e.to] = x;
          ed[e.to] = i;
          if (!inque[e.to]) {
            inque[e.to] = true;
            que.push(e.to);
          }
        }
      }
    }
    if (dist[t] == kInf) break;
    int f = kInf;
    for (int x = t; x != s; x = pv[x]) f = min(f, g[pv[x]][ed[x
    ]].cap);
    for (int x = t; x != s; x = pv[x]) {
      Edge &e = g[pv[x]][ed[x]];
      e.cap -= f;
      g[e.to][e.rev].cap += f;
    }
    flow += f;
    cost += f * dist[t];
  }
  return make_pair(flow, cost);
}
```

2.3 Gomory-Hu Tree

```
int g[maxn];
vector<edge> GomoryHu(int n){
  vector<edge> rt;
  for(int i=1;i<=n;++i)g[i]=1;
  for(int i=2;i<=n;++i){
    int t=g[i];
    flow.reset(); // clear flows on all edge
    rt.push_back({i,t,flow(i,t)});
    flow.walk(i); // bfs points that connected to i (use edges
     not fully flow)
    for(int j=i+1;j<=n;++j){
      if(g[j]==t && flow.connect(j))g[j]=i; // check if i can
     reach j
    }
  }
  return rt;
}
```

2.4 Stoer–Wagner Minimum Cut

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

2.5 Kuhn–Munkres Algorithm

```
int64_t KuhnMunkres(vector<vector<int>> W) {
  int N = W.size();
  vector<int> fl(N, -1), fr(N, -1), hr(N), hl(N);
  for (int i = 0; i < N; ++i) {
    hl[i] = *max_element(W[i].begin(), W[i].end());
  }
  auto Bfs = [&](int s) {
    vector<int> slk(N, kInf), pre(N);
    vector<bool> vl(N, false), vr(N, false);
    queue<int> que;
    que.push(s);
    vr[s] = true;
    auto Check = [&](int x) -> bool {
      if (vl[x] = true, fl[x] != -1) {
        que.push(fl[x]);
        return vr[fl[x]] = true;
```

```
      }
      while (x != -1) swap(x, fr[fl[x] = pre[x]]);
      return false;
    };
    while (true) {
      while (!que.empty()) {
        int y = que.front(); que.pop();
        for (int x = 0, d = 0; x < N; ++x) {
          if (!vl[x] && slk[x] >= (d = hl[x] + hr[y] - W[x][y])
    ) {
            if (pre[x] = y, d) slk[x] = d;
            else if (!Check(x)) return;
          }
        }
      }
      int d = kInf;
      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;
      }
    }
  };
  for (int i = 0; i < N; ++i) Bfs(i);
  int64_t res = 0;
  for (int i = 0; i < N; ++i) res += W[i][fl[i]];
  return res;
}
```

2.6 Maximum Matching on General Graph

```
namespace matching {
int fa[kN], pre[kN], match[kN], s[kN], v[kN];
vector<int> g[kN];
queue<int> q;
void Init(int n) {
  for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
  for (int i = 0; i < n; ++i) g[i].clear();
}
void AddEdge(int u, int v) {
  g[u].push_back(v);
  g[v].push_back(u);
}
int Find(int u) {
  return u == fa[u] ? u : fa[u] = Find(fa[u]);
}
int LCA(int x, int y, int n) {
  static int tk = 0;
  tk++;
  x = Find(x), y = Find(y);
  for (; ; swap(x, y)) {
    if (x != n) {
      if (v[x] == tk) return x;
      v[x] = tk;
      x = Find(pre[match[x]]);
    }
  }
}
void Blossom(int x, int y, int l) {
  while (Find(x) != l) {
    pre[x] = y, y = match[x];
    if (s[y] == 1) q.push(y), s[y] = 0;
    if (fa[x] == x) fa[x] = l;
    if (fa[y] == y) fa[y] = l;
    x = pre[y];
  }
}
bool Bfs(int r, int n) {
  for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
  while (!q.empty()) q.pop();
  q.push(r);
  s[r] = 0;
  while (!q.empty()) {
    int x = q.front(); q.pop();
    for (int u : g[x]) {
      if (s[u] == -1) {
        pre[u] = x, s[u] = 1;
        if (match[u] == n) {
          for (int a = u, b = x, last; b != n; a = last, b =
    pre[a])
            last = match[b], match[b] = a, match[a] = b;
          return true;
        }
        q.push(match[u]);
        s[match[u]] = 0;
      } else if (!s[u] && Find(u) != Find(x)) {
        int l = LCA(u, x, n);
        Blossom(x, u, l);
        Blossom(u, x, l);
      }
    }
  }
  return false;
}
int Solve(int n) {
  int res = 0;
  for (int x = 0; x < n; ++x) {
    if (match[x] == n) res += Bfs(x, n);
  }
  return res;
}}
```

2.7 Maximum Weighted Matching on General Graph

```
struct WeightGraph {
  static const int inf = INT_MAX;
  static const int maxn = 514;
  struct edge {
    int u, v, w;
    edge(){}
    edge(int u, int v, int w): u(u), v(v), w(w) {}
  };
  int n, n_x;
  edge g[maxn * 2][maxn * 2];
  int lab[maxn * 2];
  int match[maxn * 2], slack[maxn * 2], st[maxn * 2], pa[maxn *
    2];
  int flo_from[maxn * 2][maxn + 1], S[maxn * 2], vis[maxn * 2];
  vector<int> flo[maxn * 2];
  queue<int> q;
  int e_delta(const edge &e) { return lab[e.u] + lab[e.v] - g[e
    .u][e.v].w * 2; }
  void update_slack(int u, int x) { if (!slack[x] || e_delta(g[
    u][x]) < e_delta(g[slack[x]][x])) slack[x] = u; }
  void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
      if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
        update_slack(u, x);
  }
  void q_push(int x) {
    if (x <= n) q.push(x);
    else for (size_t i = 0; i < flo[x].size(); i++) q_push(flo[
    x][i]);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n) for (size_t i = 0; i < flo[x].size(); ++i)
    set_st(flo[x][i], b);
  }
  int get_pr(int b, int xr) {
    int pr = find(flo[b].begin(), flo[b].end(), xr) - flo[b].
    begin();
    if (pr % 2 == 1) {
      reverse(flo[b].begin() + 1, flo[b].end());
      return (int)flo[b].size() - pr;
    }
    return pr;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i < pr; ++i) set_match(flo[u][i], flo[u][i
    ^ 1]);
    set_match(xr, v);
    rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].end());
  }
  void augment(int u, int v) {
    for (; ; ) {
      int xnv = st[match[u]];
      set_match(u, v);
      if (!xnv) return;
      set_match(xnv, st[pa[xnv]]);
```

```
      u = st[pa[xnv]], v = xnv;
    }
  }
  int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
      if (u == 0) continue;
      if (vis[u] == t) return u;
      vis[u] = t;
      u = st[match[u]];
      if (u) u = st[pa[u]];
    }
    return 0;
  }
  void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[match[x]]),
     q_push(y);
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[match[x]]),
     q_push(y);
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      for (int x = 1; x <= n_x; ++x)
        if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][
     x]))
          g[b][x] = g[xs][x], g[x][b] = g[x][xs];
      for (int x = 1; x <= n; ++x)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
  }
  void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i)
      set_st(flo[b][i], flo[b][i]);
    int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
      int xs = flo[b][i], xns = flo[b][i + 1];
      pa[xs] = g[xns][xs].u;
      S[xs] = 1, S[xns] = 0;
      slack[xs] = 0, set_slack(xns);
      q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
  }
  bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
      pa[v] = e.u, S[v] = 1;
      int nu = st[match[v]];
      slack[v] = slack[nu] = 0;
      S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
      int lca = get_lca(u, v);
      if (!lca) return augment(u,v), augment(v,u), true;
      else add_blossom(u, lca, v);
    }
    return false;
  }
  bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
      if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0, q_push(
     x);
    if (q.empty()) return false;
    for (; ; ) {
      while (q.size()) {
        int u = q.front(); q.pop();
        if (S[st[u]] == 1) continue;
        for (int v = 1; v <= n; ++v)
          if (g[u][v].w > 0 && st[u] != st[v]) {
            if (e_delta(g[u][v]) == 0) {
              if (on_found_edge(g[u][v])) return true;
            } else update_slack(u, st[v]);
          }
      }
      int d = inf;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x]) {
          if (S[x] == -1) d = min(d, e_delta(g[slack[x]][x]));
          else if (S[x] == 0) d = min(d, e_delta(g[slack[x]][x
     ]) / 2);
        }
      for (int u = 1; u <= n; ++u) {
        if (S[st[u]] == 0) {
          if (lab[u] <= d) return 0;
          lab[u] -= d;
        } else if (S[st[u]] == 1) lab[u] += d;
      }
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b) {
          if (S[st[b]] == 0) lab[b] += d * 2;
          else if (S[st[b]] == 1) lab[b] -= d * 2;
        }
      q = queue<int>();
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x] && st[slack[x]] != x &&
     e_delta(g[slack[x]][x]) == 0)
          if (on_found_edge(g[slack[x]][x])) return true;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S[b] == 1 && lab[b] == 0)
     expand_blossom(b);
    }
    return false;
  }
  pair<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v) {
        flo_from[u][v] = (u == v ? u : 0);
        w_max = max(w_max, g[u][v].w);
      }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
      if (match[u] && match[u] < u)
        tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
  }
  void add_edge(int ui, int vi, int wi) { g[ui][vi].w = g[vi][
     ui].w = wi; }
  void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v)
        g[u][v] = edge(u, v, 0);
  }
};
```

2.8 Minimum Cost Circulation

```
struct Edge { int to, cap, rev, cost; };
vector<Edge> g[kN];
int dist[kN], pv[kN], ed[kN];
bool mark[kN];
int NegativeCycle(int n) {
  memset(mark, false, sizeof(mark));
  memset(dist, 0, sizeof(dist));
  int upd = -1;
  for (int i = 0; i <= n; ++i) {
    for (int j = 0; j < n; ++j) {
      int idx = 0;
      for (auto &e : g[j]) {
        if (e.cap > 0 && dist[e.to] > dist[j] + e.cost) {
          dist[e.to] = dist[j] + e.cost;
          pv[e.to] = j, ed[e.to] = idx;
          if (i == n) {
```

```
            upd = j;
            while (!mark[upd]) mark[upd] = true, upd = pv[upd];
            return upd;
          }
        }
        idx++;
      }
    }
  }
  return -1;
}
int Solve(int n) {
  int rt = -1, ans = 0;
  while ((rt = NegativeCycle(n)) >= 0) {
    memset(mark, false, sizeof(mark));
    vector<pair<int, int>> cyc;
    while (!mark[rt]) {
      cyc.emplace_back(pv[rt], ed[rt]);
      mark[rt] = true;
      rt = pv[rt];
    }
    reverse(cyc.begin(), cyc.end());
    int cap = kInf;
    for (auto &i : cyc) {
      auto &e = g[i.first][i.second];
      cap = min(cap, e.cap);
    }
    for (auto &i : cyc) {
      auto &e = g[i.first][i.second];
      e.cap -= cap;
      g[e.to][e.rev].cap += cap;
      ans += e.cost * cap;
    }
  }
  return ans;
}
```

2.9 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.

- 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
$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x\bar{y} + \sum_{xyx'y'} c_{xyx'y'}(x\bar{y} + x'\bar{y'})$$
  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/stdc++.h>
#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
  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

```
namespace lichao {
struct line {
  long long a, b;
  line(): a(0), b(0) {}
  line(long long a, long long b): a(a), b(b) {}
  long long operator()(int x) const { return a * x + b; }
};
line st[maxc * 4];
int sz, lc[maxc * 4], rc[maxc * 4];
int gnode() {
  st[sz] = line(1e9, 1e9);
  lc[sz] = -1, rc[sz] = -1;
  return sz++;
}
void init() {
  sz = 0;
}
void add(int l, int r, line tl, int o) {
  bool lcp = st[o](l) > tl(l);
  bool mcp = st[o]((l + r) / 2) > tl((l + r) / 2);
  if (mcp) swap(st[o], tl);
  if (r - l == 1) return;
  if (lcp != mcp) {
    if (lc[o] == -1) lc[o] = gnode();
    add(l, (l + r) / 2, tl, lc[o]);
  } else {
    if (rc[o] == -1) rc[o] = gnode();
    add((l + r) / 2, r, tl, rc[o]);
  }
}
long long query(int l, int r, int x, int o) {
  if (r - l == 1) return st[o](x);
  if (x < (l + r) / 2) {
    if (lc[o] == -1) return st[o](x);
    return min(st[o](x), query(l, (l + r) / 2, x, lc[o]));
  } else {
    if (rc[o] == -1) return st[o](x);
    return min(st[o](x), query((l + r) / 2, r, x, rc[o]));
  }
}}
```

3.3 Link-Cut Tree

```
struct node {
  node *ch[2], *fa, *pfa;
  int sum, v, rev, id;
  node(int s, int id): id(id), v(s), sum(s), rev(0), fa(nullptr
    ), pfa(nullptr) {
    ch[0] = nullptr;
    ch[1] = nullptr;
  }
  int relation() {
    return this == fa->ch[0] ? 0 : 1;
  }
  void push() {
    if (!rev) return;
    swap(ch[0], ch[1]);
    if (ch[0]) ch[0]->rev ^= 1;
    if (ch[1]) ch[1]->rev ^= 1;
    rev = 0;
  }
  void pull() {
    sum = v;
    if (ch[0]) sum += ch[0]->sum;
    if (ch[1]) sum += ch[1]->sum;
  }
  void rotate() {
    if (fa->fa) fa->fa->push();
    fa->push(), push(), swap(pfa, fa->pfa);
    int d = relation();
    node *t = fa;
    if (t->fa) t->fa->ch[t->relation()] = this;
    fa = t->fa, t->ch[d] = ch[d ^ 1];
    if (ch[d ^ 1]) ch[d ^ 1]->fa = t;
    ch[d ^ 1] = t, t->fa = this;
    t->pull(), pull();
  }
  void splay() {
    while (fa) {
      if (!fa->fa) {
        rotate();
        continue;
      }
      fa->fa->push(), fa->push();
      if (relation() == fa->relation()) fa->rotate(), rotate();
      else rotate(), rotate();
    }
  }
  void evert() { access(), splay(), rev ^= 1; }
  void expose() {
    splay(), push();
    if (ch[1]) {
      ch[1]->fa = nullptr, ch[1]->pfa = this;
      ch[1] = nullptr, pull();
    }
  }
  bool splice() {
    splay();
    if (!pfa) return false;
    pfa->expose(), pfa->ch[1] = this, fa = pfa;
    pfa = nullptr, fa->pull();
    return true;
  }
  void access() {
    expose();
    while (splice());
  }
  int query() { return sum; }
};

namespace lct {
node *sp[maxn];
void make(int u, int v) {
  // create node with id u and value v
  sp[u] = new node(v, u);
}
void link(int u, int v) {
  // u become v's parent
  sp[v]->evert();
  sp[v]->pfa = sp[u];
}
void cut(int u, int v) {
  // u was v's parent
  sp[u]->evert();
  sp[v]->access(), sp[v]->splay(), sp[v]->push();
  sp[v]->ch[0]->fa = nullptr;
  sp[v]->ch[0] = nullptr;
  sp[v]->pull();
}
void modify(int u, int v) {
  sp[u]->splay();
  sp[u]->v = v;
  sp[u]->pull();
}
int query(int u, int v) {
  sp[u]->evert(), sp[v]->access(), sp[v]->splay();
  return sp[v]->query();
}
int find(int u) {
  sp[u]->access();
  sp[u]->splay();
  node *p = sp[u];
  while (true) {
    p->push();
    if (p->ch[0]) p = p->ch[0];
    else break;
  }
  return p->id;
}}
```

4 Graph

4.1 Heavy-Light Decomposition

```
void dfs(int x, int p) {
  dep[x] = ~p ? dep[p] + 1 : dep[x];
  sz[x] = 1;
  to[x] = -1;
  fa[x] = p;
  for (const int &u : g[x]) {
    if (u == p) continue;
    dfs(u, x);
    sz[x] += sz[u];
    if (to[x] == -1 || sz[to[x]] < sz[u]) to[x] = u;
  }
}
void hld(int x, int t) {
  static int tk = 0;
  fr[x] = t;
  dfn[x] = tk++;
  if (!~to[x]) return;
  hld(to[x], t);
  for (const int &u : g[x]) {
    if (u == fa[x] || u == to[x]) continue;
    hld(u, u);
  }
}
vector<pair<int, int>> get(int x, int y) {
  int fx = fr[x], fy = fr[y];
  vector<pair<int, int>> res;
  while (fx != fy) {
    if (dep[fx] < dep[fy]) {
      swap(fx, fy);
      swap(x, y);
    }
    res.emplace_back(dfn[fx], dfn[x] + 1);
    x = fa[fx];
    fx = fr[x];
  }
  res.emplace_back(min(dfn[x], dfn[y]), max(dfn[x], dfn[y]) +
     1);
  int lca = (dep[x] < dep[y] ? x : y);
  return res;
}
```

4.2 Centroid Decomposition

```
void get_center(int now) {
  v[now] = true; vtx.push_back(now);
  sz[now] = 1; mx[now] = 0;
  for (int u : G[now]) if (!v[u]) {
    get_center(u);
    mx[now] = max(mx[now], sz[u]);
    sz[now] += sz[u];
  }
}
void get_dis(int now, int d, int len) {
  dis[d][now] = cnt;
  v[now] = true;
  for (auto u : G[now]) if (!v[u.first]) {
    get_dis(u, d, len + u.second);
  }
}
```

```cpp
void dfs(int now, int fa, int d) {
  get_center(now);
  int c = -1;
  for (int i : vtx) {
    if (max(mx[i], (int)vtx.size() - sz[i]) <= (int)vtx.size()
     / 2) c = i;
    v[i] = false;
  }
  get_dis(c, d, 0);
  for (int i : vtx) v[i] = false;
  v[c] = true; vtx.clear();
  dep[c] = d; p[c] = fa;
  for (auto u : G[c]) if (u.first != fa && !v[u.first]) {
    dfs(u.first, c, d + 1);
  }
}
```

4.3 Minimum Mean Cycle

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

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

4.4 Minimum Steiner Tree

```cpp
namespace steiner {
// Minimum Steiner Tree - O(N * 3^K + N^2 * 2^K)
// z[i] = the weight of the i-th vertex
const int maxn = 64, maxk = 10;
const int inf = 1e9;
int w[maxn][maxn], z[maxn], dp[1 << maxk][maxn], off[maxn];
void init(int n) {
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) w[i][j] = inf;
    z[i] = 0;
    w[i][i] = 0;
  }
}
void add_edge(int x, int y, int d) {
  w[x][y] = min(w[x][y], d);
  w[y][x] = min(w[y][x], d);
}
void build(int n) {
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      w[i][j] += z[i];
      if (i != j) w[i][j] += z[j];
    }
  }
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) w[i][j] = min(w[i][j], w[i][k
     ] + w[k][j] - z[k]);
    }
  }
}
int solve(int n, vector<int> mark) {
  build(n);
  int k = (int)mark.size();
  assert(k < maxk);
  for (int s = 0; s < (1 << k); ++s) {
    for (int i = 0; i < n; ++i) dp[s][i] = inf;
  }
  for (int i = 0; i < n; ++i) dp[0][i] = 0;
  for (int s = 1; s < (1 << k); ++s) {
    if (__builtin_popcount(s) == 1) {
      int x = __builtin_ctz(s);
      for (int i = 0; i < n; ++i) dp[s][i] = w[mark[x]][i];
      continue;
    }
    for (int i = 0; i < n; ++i) {
      for (int sub = s & (s - 1); sub; sub = s & (sub - 1)) {
        dp[s][i] = min(dp[s][i], dp[sub][i] + dp[s ^ sub][i] -
     z[i]);
      }
    }
    for (int i = 0; i < n; ++i) {
      off[i] = inf;
      for (int j = 0; j < n; ++j) off[i] = min(off[i], dp[s][j]
     + w[j][i] - z[j]);
    }
    for (int i = 0; i < n; ++i) dp[s][i] = min(dp[s][i], off[i
     ]);
  }
  int res = inf;
  for (int i = 0; i < n; ++i) res = min(res, dp[(1 << k) - 1][i
     ]);
  return res;
}}
```

4.5 Directed Minimum Spanning Tree

```cpp
template <typename T> struct DMST {
  T g[maxn][maxn], fw[maxn];
  int n, fr[maxn];
  bool vis[maxn], inc[maxn];
  void clear() {
    for(int i = 0; i < maxn; ++i) {
      for(int j = 0; j < maxn; ++j) g[i][j] = inf;
      vis[i] = inc[i] = false;
    }
  }
  void addedge(int u, int v, T w) {
    g[u][v] = min(g[u][v], w);
  }
  T operator()(int root, int _n) {
    n = _n;
    if (dfs(root) != n) return -1;
    T ans = 0;
    while (true) {
      for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
      for (int i = 1; i <= n; ++i) if (!inc[i]) {
        for (int j = 1; j <= n; ++j) {
          if (!inc[j] && i != j && g[j][i] < fw[i]) {
            fw[i] = g[j][i];
            fr[i] = j;
          }
        }
      }
      int x = -1;
      for (int i = 1; i <= n; ++i) if (i != root && !inc[i]) {
        int j = i, c = 0;
        while (j != root && fr[j] != i && c <= n) ++c, j = fr[j
     ];
        if (j == root || c > n) continue;
        else { x = i; break; }
      }
      if (!~x) {
        for (int i = 1; i <= n; ++i) if (i != root && !inc[i])
     ans += fw[i];
        return ans;
      }
      int y = x;
      for (int i = 1; i <= n; ++i) vis[i] = false;
      do { ans += fw[y]; y = fr[y]; vis[y] = inc[y] = true; }
     while (y != x);
      inc[x] = false;
      for (int k = 1; k <= n; ++k) if (vis[k]) {
        for (int j = 1; j <= n; ++j) if (!vis[j]) {
          if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
          if (g[j][k] < inf && g[j][k] - fw[k] < g[j][x]) g[j][
     x] = g[j][k] - fw[k];
        }
      }
    }
    return ans;
  }
  int dfs(int now) {
```

```
    int r = 1;
    vis[now] = true;
    for (int i = 1; i <= n; ++i) if (g[now][i] < inf && !vis[i
     ]) r += dfs(i);
    return r;
  }
};
```

4.6 Maximum Clique

```
struct MaxClique {
  // change to bitset for n > 64.
  int n, deg[maxn];
  uint64_t adj[maxn], ans;
  vector<pair<int, int>> edge;
  void init(int n_) {
    n = n_;
    fill(adj, adj + n, 0ull);
    fill(deg, deg + n, 0);
    edge.clear();
  }
  void add_edge(int u, int v) {
    edge.emplace_back(u, v);
    ++deg[u], ++deg[v];
  }
  vector<int> operator()() {
    vector<int> ord(n);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int u, int v) { return deg
     [u] < deg[v]; });
    vector<int> id(n);
    for (int i = 0; i < n; ++i) id[ord[i]] = i;
    for (auto e : edge) {
      int u = id[e.first], v = id[e.second];
      adj[u] |= (1ull << v);
      adj[v] |= (1ull << u);
    }
    uint64_t r = 0, p = (1ull << n) - 1;
    ans = 0;
    dfs(r, p);
    vector<int> res;
    for (int i = 0; i < n; ++i) {
      if (ans >> i & 1) res.push_back(ord[i]);
    }
    return res;
  }
#define pcount __builtin_popcountll
  void dfs(uint64_t r, uint64_t p) {
    if (p == 0) {
      if (pcount(r) > pcount(ans)) ans = r;
      return;
    }
    if (pcount(r | p) <= pcount(ans)) return;
    int x = __builtin_ctzll(p & -p);
    uint64_t c = p & ~adj[x];
    while (c > 0) {
      // bitset._Find_first(); bitset._Find_next();
      x = __builtin_ctzll(c & -c);
      r |= (1ull << x);
      dfs(r, p & adj[x]);
      r &= ~(1ull << x);
      p &= ~(1ull << x);
      c ^= (1ull << x);
    }
  }
};
```

4.7 Tarjan's Algorithm

```
void dfs(int x, int p) {
  dfn[x] = low[x] = tk++;
  int ch = 0;
  st.push(x); // bridge
  for (auto e : g[x]) if (e.first != p) {
    if (!ins[e.second]) { // articulation point
      st.push(e.second);
      ins[e.second] = true;
    }
    if (~dfn[e.first]) {
      low[x] = min(low[x], dfn[e.first]);
      continue;
    }
    dfs(u.first, x);
    if (low[u.first] >= low[x]) { // articulation point
      cut[x] = true;
      while (true) {
        int z = st.top(); st.pop();
        bcc[z] = sz;
        if (z == e.second) break;
      }
      sz++;
    }
  }
  if (ch == 1 && p == -1) cut[x] = false;
  if (dfn[x] == low[x]) { // bridge
    while (true) {
      int z = st.top(); st.pop();
      bcc[z] = sz;
      if (z == x) break;
    }
  }
}
```

4.8 Dominator Tree

```
vector<int> BuildDominatorTree(vector<vector<int>> g, int s) {
  int N = g.size();
  vector<vector<int>> rdom(N), r(N);
  vector<int> dfn(N, -1), rev(N, -1), fa(N, -1), sdom(N, -1),
     dom(N, -1), val(N, -1), rp(N, -1);
  int stamp = 0;
  auto Dfs = [&](auto dfs, int x) -> void {
    rev[dfn[x] = stamp] = x;
    fa[stamp] = sdom[stamp] = val[stamp] = stamp;
    stamp++;
    for (int u : g[x]) {
      if (dfn[u] == -1) {
        dfs(dfs, u);
        rp[dfn[u]] = dfn[x];
      }
      r[dfn[u]].push_back(dfn[x]);
    }
  };
  function<int(int, int)> Find = [&](int x, int c) {
    if (fa[x] == x) return c ? -1 : x;
    int p = Find(fa[x], 1);
    if (p == -1) return c ? fa[x] : val[x];
    if (sdom[val[x]] > sdom[val[fa[x]]]) val[x] = val[fa[x]];
    fa[x] = p;
    return c ? p : val[x];
  };
  auto Merge = [&](int x, int y) { fa[x] = y; };
  Dfs(Dfs, s);
  for (int i = stamp - 1; i >= 0; --i) {
    for (int u : r[i]) sdom[i] = min(sdom[i], sdom[Find(u, 0)])
     ;
    if (i) 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) Merge(i, rp[i]);
  }
  vector<int> res(N, -2);
  res[s] = -1;
  for (int i = 1; i < stamp; ++i) {
    if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
  }
  for (int i = 1; i < stamp; ++i) res[rev[i]] = rev[dom[i]];
  return res;
}
```

4.9 Virtual Tree

```
void VirtualTree(vector<int> v) {
  v.push_back(0);
  sort(v.begin(), v.end(), [&](int i, int j) { return dfn[i] <
     dfn[j]; });
  v.resize(unique(v.begin(), v.end()) - v.begin());
  vector<int> stk;
  for (int u : v) {
    if (stk.empty()) {
      stk.push_back(u);
      continue;
    }
    int p = GetLCA(u, stk.back());
    if (p != stk.back()) {
      while (stk.size() >= 2 && dep[p] <= dep[stk[stk.size() -
     2]]) {
        int x = stk.back();
        stk.pop_back();
        AddEdge(x, stk.back());
      }
```

```
      if (stk.back() != p) {
        AddEdge(stk.back(), p);
        stk.pop_back();
        stk.push_back(p);
      }
    }
    stk.push_back(u);
  }
  for (int i = 0; i + 1 < stk.size(); ++i) AddEdge(stk[i], stk[
     i + 1]);
}
```

4.10 Vizing's Theorem

```
namespace vizing { // returns edge coloring in adjacent matrix
     G. 1 - based
int C[kN][kN], G[kN][kN];
void clear(int N) {
  for (int i = 0; i <= N; i++) {
    for (int j = 0; j <= N; j++) C[i][j] = G[i][j] = 0;
  }
}
void solve(vector<pair<int, int>> &E, int N, int M) {
  int X[kN] = {}, a;
  auto update = [&](int u) {
    for (X[u] = 1; C[u][X[u]]; X[u]++);
  };
  auto color = [&](int u, int v, int c) {
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v, C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if (p) X[u] = X[v] = p;
    else update(u), update(v);
    return p;
  };
  auto flip = [&](int u, int c1, int c2) {
    int p = C[u][c1];
    swap(C[u][c1], C[u][c2]);
    if (p) G[u][p] = G[p][u] = c2;
    if (!C[u][c1]) X[u] = c1;
    if (!C[u][c2]) X[u] = c2;
    return p;
  };
  for (int i = 1; i <= N; i++) X[i] = 1;
  for (int t = 0; t < E.size(); t++) {
    int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c
     = c0, d;
    vector<pair<int, int>> L;
    int vst[kN] = {};
    while (!G[u][v0]) {
      L.emplace_back(v, d = X[v]);
      if (!C[v][c]) for (a = (int)L.size() - 1; a >= 0; a--) c
     = color(u, L[a].first, c);
      else if (!C[u][d]) for (a = (int)L.size() - 1; a >= 0; a
     --) color(u, L[a].first, L[a].second);
      else if (vst[d]) break;
      else vst[d] = 1, v = C[u][d];
    }
    if (!G[u][v0]) {
      for (; v; v = flip(v, c, d), swap(c, d));
      if (C[u][c0]) {
        for (a = (int)L.size() - 2; a >= 0 && L[a].second != c;
     a--);
        for (; a >= 0; a--) color(u, L[a].first, L[a].second);
      } else t--;
    }
  }
}}
```

4.11 System of Difference Constraints

Given m constrains on n variables $x_1, x_2, \ldots, x_n$ of form $x_i - x_j \leq w$ (resp, $x_i - x_j \geq w$), connect $i \to j$ with weight w. Then connect $0 \to i$ for all i with weight 0 and find the shortest path (resp, longest path) on the graph. $dis(i)$ will be the maximum (resp, minimum) solution to x_i.

5 String

5.1 Knuth–Morris–Pratt Algorithm

```
vector<int> kmp(const string &s) {
  vector<int> f(s.size(), 0);
  // f[i] = length of the longest prefix (excluding s[0:i])
     such that it coincides with the suffix of s[0:i] of the
     same length
  // i + 1 - f[i] is the length of the smallest recurring
     period of s[0:i]
  int k = 0;
  for (int i = 1; i < (int)s.size(); ++i) {
    while (k > 0 && s[i] != s[k]) k = f[k - 1];
    if (s[i] == s[k]) ++k;
    f[i] = k;
  }
  return f;
}
vector<int> search(const string &s, const string &t) {
  // return 0-indexed occurrence of t in s
  vector<int> f = kmp(t), res;
  for (int i = 0, k = 0; i < (int)s.size(); ++i) {
    while (k > 0 && (k == (int)t.size() || s[i] != t[k])) k = f
     [k - 1];
    if (s[i] == t[k]) ++k;
    if (k == (int)t.size()) res.push_back(i - t.size() + 1);
  }
  return res;
}
```

5.2 Z Algorithm

```
int z[maxn];
// z[i] = LCP of suffix i and suffix 0
void z_function(const string& s) {
  memset(z, 0, sizeof(z));
  z[0] = (int)s.length();
  int l = 0, r = 0;
  for (int i = 1; i < s.length(); ++i) {
    z[i] = max(0, min(z[i - l], r - i + 1));
    while (i + z[i] < s.length() && s[z[i]] == s[i + z[i]]) {
      l = i; r = i + z[i];
      ++z[i];
    }
  }
}
```

5.3 Manacher's Algorithm

```
int z[maxn];
int manacher(const string& s) {
  string t = ".";
  for (int i = 0; i < s.length(); ++i) t += s[i], t += '.';
  int l = 0, r = 0, ans = 0;
  for (int i = 1; i < t.length(); ++i) {
    z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
    while (i - z[i] >= 0 && i + z[i] < t.length() && t[i - z[i
     ]] == t[i + z[i]]) ++z[i];
    if (i + z[i] > r) r = i + z[i], l = i;
  }
  for (int i = 1; i < t.length(); ++i) ans = max(ans, z[i] - 1)
     ;
  return ans;
}
```

5.4 Aho-Corasick Automaton

```
struct AC {
  static const int maxn = 1e5 + 5;
  int sz, ql, qr, root;
  int cnt[maxn], q[maxn], ed[maxn], el[maxn], ch[maxn][26], f[
     maxn];
  int gnode() {
    for (int i = 0; i < 26; ++i) ch[sz][i] = -1;
    f[sz] = -1;
    ed[sz] = 0;
    cnt[sz] = 0;
    return sz++;
  }
  void init() {
    sz = 0;
    root = gnode();
  }
  int add(const string &s) {
    int now = root;
    for (int i = 0; i < s.length(); ++i) {
      if (ch[now][s[i] - 'a'] == -1) ch[now][s[i] - 'a'] =
     gnode();
      now = ch[now][s[i] - 'a'];
    }
    ed[now] = 1;
    return now;
  }
  void build_fail() {
    ql = qr = 0; q[qr++] = root;
```

```
    while (ql < qr) {
      int now = q[ql++];
      for (int i = 0; i < 26; ++i) if (ch[now][i] != -1) {
        int p = ch[now][i], fp = f[now];
        while (fp != -1 && ch[fp][i] == -1) fp = f[fp];
        int pd = fp != -1 ? ch[fp][i] : root;
        f[p] = pd;
        el[p] = ed[pd] ? pd : el[pd];
        q[qr++] = p;
      }
    }
  }
  void build(const string &s) {
    build_fail();
    int now = root;
    for (int i = 0; i < s.length(); ++i) {
      while (now != -1 && ch[now][s[i] - 'a'] == -1) now = f[
     now];
      now = now != -1 ? ch[now][s[i] - 'a'] : root;
      ++cnt[now];
    }
    for (int i = qr - 1; i >= 0; --i) cnt[f[q[i]]] += cnt[q[i
     ]];
  }
  long long solve(int n) {
    build_fail();
    vector<vector<long long>> dp(sz, vector<long long>(n + 1,
     0));
    for (int i = 0; i < sz; ++i) dp[i][0] = 1;
    for (int i = 1; i <= n; ++i) {
      for (int j = 0; j < sz; ++j) {
        for (int k = 0; k < 2; ++k) {
          if (ch[j][k] != -1) {
            if (!ed[ch[j][k]])
              dp[j][i] += dp[ch[j][k]][i - 1];
          } else {
            int z = f[j];
            while (z != root && ch[z][k] == -1) z = f[z];
            int p = ch[z][k] == -1 ? root : ch[z][k];
            if (ch[z][k] == -1 || !ed[ch[z][k]]) dp[j][i] += dp
     [p][i - 1];
          }
        }
      }
    }
    return dp[0][n];
  }
};
```

5.5 Suffix Automaton

```
struct SAM {
  static const int maxn = 5e5 + 5;
  int nxt[maxn][26], to[maxn], len[maxn];
  int root, last, sz;
  int gnode(int x) {
    for (int i = 0; i < 26; ++i) nxt[sz][i] = -1;
    to[sz] = -1;
    len[sz] = x;
    return sz++;
  }
  void init() {
    sz = 0;
    root = gnode(0);
    last = root;
  }
  void push(int c) {
    int cur = last;
    last = gnode(len[last] + 1);
    for (; ~cur && nxt[cur][c] == -1; cur = to[cur]) nxt[cur][c
     ] = last;
    if (cur == -1) return to[last] = root, void();
    int link = nxt[cur][c];
    if (len[link] == len[cur] + 1) return to[last] = link, void
     ();
    int tlink = gnode(len[cur] + 1);
    for (; ~cur && nxt[cur][c] == link; cur = to[cur]) nxt[cur
     ][c] = tlink;
    for (int i = 0; i < 26; ++i) nxt[tlink][i] = nxt[link][i];
    to[tlink] = to[link];
    to[link] = tlink;
    to[last] = tlink;
  }
  void add(const string &s) {
    for (int i = 0; i < s.size(); ++i) push(s[i] - 'a');
  }
  bool find(const string &s) {
    int cur = root;
    for (int i = 0; i < s.size(); ++i) {
      cur = nxt[cur][s[i] - 'a'];
      if (cur == -1) return false;
    }
    return true;
  }
  int solve(const string &t) {
    int res = 0, cnt = 0;
    int cur = root;
    for (int i = 0; i < t.size(); ++i) {
      if (~nxt[cur][t[i] - 'a']) {
        ++cnt;
        cur = nxt[cur][t[i] - 'a'];
      } else {
        for (; ~cur && nxt[cur][t[i] - 'a'] == -1; cur = to[cur
     ]);
        if (~cur) cnt = len[cur] + 1, cur = nxt[cur][t[i] - 'a'
     ];
        else cnt = 0, cur = root;
      }
      res = max(res, cnt);
    }
    return res;
  }
};
```

5.6 Suffix Array

```
// sa[i]: sa[i]-th suffix is the i-th lexigraphically smallest
     suffix.
// lcp[i]: longest common prefix of suffix sa[i] and suffix sa[
     i - 1].
namespace sfx {
vector<int> Build(const string &s) {
  int n = s.size();
  vector<int> str(n * 2), sa(n * 2), c(max(n, 256) * 2), x(max(
     n, 256)), p(n), q(n * 2), t(n * 2);
  for (int i = 0; i < n; ++i) str[i] = s[i];
  auto Pre = [&](int *sa, int *c, int n, int z) {
    memset(sa, 0, sizeof(int) * n);
    memcpy(x.data(), c, sizeof(int) * z);
  };
  auto Induce = [&](int *sa, int *c, int *s, int *t, int n, int
      z) {
    memcpy(x.data() + 1, c, sizeof(int) * (z - 1));
    for (int i = 0; i < n; ++i) if (sa[i] && !t[sa[i] - 1]) sa[
     x[s[sa[i] - 1]]++] = sa[i] - 1;
    memcpy(x.data(), c, sizeof(int) * z);
    for (int i = n - 1; i >= 0; --i) if (sa[i] && t[sa[i] - 1])
      sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
  };
  auto SAIS = [&](auto self, int *s, int *sa, int *p, int *q,
     int *t, int *c, int n, int z) -> void {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n, last =
     -1;
    memset(c, 0, sizeof(int) * z);
    for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
    for (int i = 0; i < z - 1; ++i) c[i + 1] += c[i];
    if (uniq) {
      for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
      return;
    }
    for (int i = n - 2; i >= 0; --i) t[i] = (s[i] == s[i + 1] ?
      t[i + 1] : s[i] < s[i + 1]);
    Pre(sa, c, n, z);
    for (int i = 1; i <= n - 1; ++i) if (t[i] && !t[i - 1]) sa
     [--x[s[i]]] = p[q[i] = nn++] = i;
    Induce(sa, c, s, t, n, z);
    for (int i = 0; i < n; ++i) if (sa[i] && t[sa[i]] && !t[sa[
     i] - 1]) {
      bool neq = last < 0 || memcmp(s + sa[i], s + last, (p[q[
     sa[i]] + 1] - sa[i]) * sizeof(int));
      ns[q[last = sa[i]]] = nmxz += neq;
    }
    self(self, ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz +
      1);
    Pre(sa, c, n, z);
    for (int i = nn - 1; i >= 0; --i) sa[--x[s[p[nsa[i]]]]] = p
     [nsa[i]];
    Induce(sa, c, s, t, n, z);
  };
  SAIS(SAIS, str.data(), sa.data(), p.data(), q.data(), t.data
     (), c.data(), n + 1, 256);
  return vector<int>(sa.begin() + 1, sa.begin() + n + 1);
```

```
}
vector<int> BuildLCP(const vector<int> &sa, const string &s) {
  int n = s.size();
  vector<int> lcp(n), rev(n);
  for (int i = 0; i < n; ++i) rev[sa[i]] = i;
  for (int i = 0, ptr = 0; i < n; ++i) {
    if (!rev[i]) {
      ptr = 0;
      continue;
    }
    while (i + ptr < n && s[i + ptr] == s[sa[rev[i] - 1] + ptr
     ]) ptr++;
    lcp[rev[i]] = ptr ? ptr-- : 0;
  }
  return lcp;
}} // namespace sfx
```

5.7 Palindromic Tree

```
struct PalindromicTree {
  int link[kN], len[kN], dp[kN], nxt[kN][26], sz, sf;
  int gnode(int l, int fl = -1) {
    len[sz] = l;
    link[sz] = fl;
    fill(nxt[sz], nxt[sz] + 26, -1);
    return sz++;
  }
  void Init() {
    sz = 0;
    sf = 1;
    gnode(-1, 0);
    gnode(0, 0);
  }
  void Push(const string &s, int pos) {
    int cur = sf, z = s[pos] - 'a';
    while (pos - 1 - len[cur] < 0 || s[pos - 1 - len[cur]] != s
     [pos]) cur = link[cur];
    if (nxt[cur][z] != -1) {
      sf = nxt[cur][z];
    } else {
      int ch = gnode(len[cur] + 2);
      nxt[cur][z] = sf = ch;
      if (len[ch] == 1) {
        link[ch] = 1;
      } else {
        cur = link[cur];
        while (pos - 1 - len[cur] < 0 || s[pos - 1 - len[cur]]
     != s[pos]) cur = link[cur];
        link[ch] = nxt[cur][z];
      }
    }
    dp[sf] += 1;
  }
  long long Build(const string &s) {
    for (int i = 0; i < s.size(); ++i) Push(s, i);
    for (int i = sz - 1; i >= 0; --i) dp[link[i]] += dp[i];
    long long res = 0;
    for (int i = 0; i < sz; ++i) res = max(res, 1LL * dp[i] *
     len[i]);
    return res;
  }
} plt;
```

5.8 Circular LCS

```
string s1, s2;
int n, m;
int dp[kN * 2][kN];
int nxt[kN * 2][kN];
void reroot(int px) {
  int py = 1;
  while (py <= m && nxt[px][py] != 2) py++;
  nxt[px][py] = 1;
  while (px < 2 * n && py < m) {
    if (nxt[px + 1][py] == 3) px++, nxt[px][py] = 1;
    else if (nxt[px + 1][py + 1] == 2) px++, py++, nxt[px][py]
     = 1;
    else py++;
  }
  while (px < 2 * n && nxt[px + 1][py] == 3) px++, nxt[px][py]
   = 1;
}
int track(int x, int y, int e) { // use this routine to find
     LCS as string
  int ret = 0;
  while (y != 0 && x != e) {
    if (nxt[x][y] == 1) y--;
    else if (nxt[x][y] == 2) ret += (s1[x] == s2[y]), x--, y--;
    else if (nxt[x][y] == 3) x--;
  }
  return ret;
}
int solve(string a, string b) {
  n = a.size(), m = b.size();
  s1 = "#" + a + a, s1 = '#' + b;
  for (int i = 0; i <= 2 * n; i++) {
    for (int j = 0; j <= m; j++) {
      if (j == 0) { nxt[i][j] = 3; continue; }
      if (i == 0) { nxt[i][j] = 1; continue; }
      dp[i][j] = -1;
      if (dp[i][j] < dp[i][j - 1]) dp[i][j] = dp[i][j - 1], nxt
     [i][j] = 1;
      if (dp[i][j] < dp[i - 1][j - 1] + (s1[i] == s2[j])) dp[i
     ][j] = dp[i - 1][j - 1] + (s1[i] == s2[j]), nxt[i][j] = 2;
      if (dp[i][j] < dp[i - 1][j]) dp[i][j] = dp[i - 1][j], nxt
     [i][j] = 3;
    }
  }
  int ret = dp[n][m];
  for (int i = 1; i < n; i++) reroot(i), ret = max(ret, track(n
     + i, m, i));
  return ret;
}
```

5.9 Lexicographically Smallest Rotation

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

6 Math

6.1 Fast Fourier Transform

```
namespace fft {
struct cplx {
  double re, im;
  cplx(): re(0), im(0) {}
  cplx(double r, double i): re(r), im(i) {}
  cplx operator+(const cplx &rhs) const { return cplx(re + rhs.
     re, im + rhs.im); }
  cplx operator-(const cplx &rhs) const { return cplx(re - rhs.
     re, im - rhs.im); }
  cplx operator*(const cplx &rhs) const { return cplx(re * rhs.
     re - im * rhs.im, re * rhs.im + im * rhs.re); }
  cplx conj() const { return cplx(re, -im); }
};
const double pi = acos(-1);
cplx omega[maxn + 1];
void prefft() {
  for (int i = 0; i <= maxn; ++i)
    omega[i] = cplx(cos(2 * pi * i / maxn), sin(2 * pi * i /
     maxn));
}
void bitrev(vector<cplx> &v, int n) {
  int z = __builtin_ctz(n) - 1;
  for (int i = 0; i < n; ++i) {
    int x = 0;
    for (int j = 0; (1 << j) < n; ++j) x ^= (i >> j & 1) << (z
     - j);
    if (x > i) swap(v[x], v[i]);
  }
}
void fft(vector<cplx> &v, int n) {
  bitrev(v, n);
  for (int s = 2; s <= n; s <<= 1) {
    int z = s >> 1;
    for (int i = 0; i < n; i += s) {
      for (int k = 0; k < z; ++k) {
        cplx x = v[i + z + k] * omega[maxn / s * k];
        v[i + z + k] = v[i + k] - x;
```

```
        v[i + k] = v[i + k] + x;
      }
    }
  }
}
void ifft(vector<cplx> &v, int n) {
  fft(v, n);
  reverse(v.begin() + 1, v.end());
  for (int i = 0; i < n; ++i) v[i] = v[i] * cplx(1. / n, 0);
}
vector<long long> convolution(const vector<int> &a, const
     vector<int> &b) {
  // Should be able to handle N <= 10^5, C <= 10^4
  int sz = 1;
  while (sz < a.size() + b.size() - 1) sz <<= 1;
  vector<cplx> v(sz);
  for (int i = 0; i < sz; ++i) {
    double re = i < a.size() ? a[i] : 0;
    double im = i < b.size() ? b[i] : 0;
    v[i] = cplx(re, im);
  }
  fft(v, sz);
  for (int i = 0; i <= sz / 2; ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (v[i] + v[j].conj()) * (v[i] - v[j].conj()) * cplx
     (0, -0.25);
    if (j != i) v[j] = (v[j] + v[i].conj()) * (v[j] - v[i].conj
     ()) * cplx(0, -0.25);
    v[i] = x;
  }
  ifft(v, sz);
  vector<long long> c(sz);
  for (int i = 0; i < sz; ++i) c[i] = round(v[i].re);
  return c;
}
vector<int> convolution_mod(const vector<int> &a, const vector<
     int> &b, int p) {
  int sz = 1;
  while (sz < (int)a.size() + (int)b.size() - 1) sz <<= 1;
  vector<cplx> fa(sz), fb(sz);
  for (int i = 0; i < (int)a.size(); ++i)
    fa[i] = cplx(a[i] & ((1 << 15) - 1), a[i] >> 15);
  for (int i = 0; i < (int)b.size(); ++i)
    fb[i] = cplx(b[i] & ((1 << 15) - 1), b[i] >> 15);
  fft(fa, sz), fft(fb, sz);
  double r = 0.25 / sz;
  cplx r2(0, -1), r3(r, 0), r4(0, -r), r5(0, 1);
  for (int i = 0; i <= (sz >> 1); ++i) {
    int j = (sz - i) & (sz - 1);
    cplx a1 = (fa[i] + fa[j].conj());
    cplx a2 = (fa[i] - fa[j].conj()) * r2;
    cplx b1 = (fb[i] + fb[j].conj()) * r3;
    cplx b2 = (fb[i] - fb[j].conj()) * r4;
    if (i != j) {
      cplx c1 = (fa[j] + fa[i].conj());
      cplx c2 = (fa[j] - fa[i].conj()) * r2;
      cplx d1 = (fb[j] + fb[i].conj()) * r3;
      cplx d2 = (fb[j] - fb[i].conj()) * r4;
      fa[i] = c1 * d1 + c2 * d2 * r5;
      fb[i] = c1 * d2 + c2 * d1;
    }
    fa[j] = a1 * b1 + a2 * b2 * r5;
    fb[j] = a1 * b2 + a2 * b1;
  }
  fft(fa, sz), fft(fb, sz);
  vector<int> res(sz);
  for (int i = 0; i < sz; ++i) {
    long long a = round(fa[i].re);
    long long b = round(fb[i].re);
    long long c = round(fa[i].im);
    res[i] = (a + ((b % p) << 15) + ((c % p) << 30)) % p;
  }
  return res;
}}
```

6.2 Number Theoretic Transform

```
vector<int> omega;
void Init() {
  omega.resize(kN + 1);
  long long x = fpow(kRoot, (Mod - 1) / kN);
  omega[0] = 1;
  for (int i = 1; i <= kN; ++i) {
    omega[i] = 1LL * omega[i - 1] * x % kMod;
  }
}
void Transform(vector<int> &v, int n) {
  BitReverse(v, n);
  for (int s = 2; s <= n; s <<= 1) {
    int z = s >> 1;
    for (int i = 0; i < n; i += s) {
      for (int k = 0; k < z; ++k) {
        int x = 1LL * v[i + k + z] * omega[kN / s * k] % kMod;
        v[i + k + z] = (v[i + k] + kMod - x) % kMod;
        (v[i + k] += x) %= kMod;
      }
    }
  }
}
void InverseTransform(vector<int> &v, int n) {
  Transform(v, n);
  for (int i = 1; i < n / 2; ++i) swap(v[i], v[n - i]);
  const int kInv = fpow(n, kMod - 2);
  for (int i = 0; i < n; ++i) v[i] = 1LL * v[i] * inv % kMod;
}
```

6.3 NTT Prime List

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

6.4 Formal Power Series

```
Poly Inverse(Poly f) {
  int n = f.size();
  Poly q(1, fpow(f[0], kMod - 2));
  for (int s = 2;; s <<= 1) {
    if (f.size() < s) f.resize(s);
    Poly fv(f.begin(), f.begin() + s);
    Poly fq(q.begin(), q.end());
    fv.resize(s + s);
    fq.resize(s + s);
    ntt::Transform(fv, s + s);
    ntt::Transform(fq, s + s);
    for (int i = 0; i < s + s; ++i) {
      fv[i] = 1LL * fv[i] * fq[i] % kMod * fq[i] % kMod;
    }
    ntt::InverseTransform(fv, s + s);
    Poly res(s);
    for (int i = 0; i < s; ++i) {
      res[i] = kMod - fv[i];
      if (i < (s >> 1)) {
        int v = 2 * q[i] % kMod;
        (res[i] += v) >= kMod ? res[i] -= kMod : 0;
      }
    }
    q = res;
    if (s >= n) break;
  }
  q.resize(n);
  return q;
}
Poly Divide(const Poly &a, const Poly &b) {
  int n = a.size(), m = b.size(), k = 2;
  while (k < n - m + 1) k <<= 1;
  Poly ra(k), rb(k);
  for (int i = 0; i < min(n, k); ++i) ra[i] = a[n - 1 - i];
  for (int i = 0; i < min(m, k); ++i) rb[i] = b[m - 1 - i];
  auto rbi = Inverse(rb);
  auto res = Multiply(rbi, ra);
  res.resize(n - m + 1);
  reverse(res.begin(), res.end());
  return res;
}
Poly Modulo(const Poly &a, const Poly &b) {
  if (a.size() < b.size()) return a;
  auto dv = Multiply(Divide(a, b), b);
  assert(dv.size() == a.size());
  for (int i = 0; i < dv.size(); ++i) {
    dv[i] = (a[i] + kMod - dv[i]) % kMod;
  }
  while (!dv.empty() && dv.back() == 0) dv.pop_back();
  return dv;
}
Poly Derivative(const Poly &f) {
  int n = f.size();
  vector<int> res(n - 1);
```

```
  for (int i = 0; i < n - 1; ++i) {
    res[i] = 1LL * f[i + 1] * (i + 1) % kMod;
  }
  return res;
}
Poly Integral(const Poly &f) {
  int n = f.size();
  vector<int> res(n + 1);
  for (int i = 0; i < n; ++i) {
    res[i + 1] = 1LL * f[i] * fpow(i + 1, kMod - 2) % kMod;
  }
  return res;
}
Poly Evaluate(const Poly &f, const vector<int> &x) {
  if (x.empty()) return Poly();
  int n = x.size();
  vector<Poly> up(n * 2);
  for (int i = 0; i < n; ++i) up[i + n] = {kMod - x[i], 1};
  for (int i = n - 1; i > 0; --i) up[i] = Multiply(up[i * 2],
     up[i * 2 + 1]);
  vector<Poly> down(n * 2);
  down[1] = Modulo(f, up[1]);
  for (int i = 2; i < n * 2; ++i) down[i] = Modulo(down[i >>
     1], up[i]);
  vector<int> y(n);
  for (int i = 0; i < n; ++i) y[i] = down[i + n][0];
  return y;
}
Poly Interpolate(const vector<int> &x, const vector<int> &y) {
  int n = x.size();
  vector<Poly> up(n * 2);
  for (int i = 0; i < n; ++i) up[i + n] = {kMod - x[i], 1};
  for (int i = n - 1; i > 0; --i) up[i] = Multiply(up[i * 2],
     up[i * 2 + 1]);
  vector<int> a = Evaluate(Derivative(up[1]), x);
  for (int i = 0; i < n; ++i) {
    a[i] = 1LL * y[i] * fpow(a[i], kMod - 2) % kMod;
  }
  vector<Poly> down(n * 2);
  for (int i = 0; i < n; ++i) down[i + n] = {a[i]};
  for (int i = n - 1; i > 0; --i) {
    auto lhs = Multiply(down[i * 2], up[i * 2 + 1]);
    auto rhs = Multiply(down[i * 2 + 1], up[i * 2]);
    assert(lhs.size() == rhs.size());
    down[i].resize(lhs.size());
    for (int j = 0; j < lhs.size(); ++j) {
      down[i][j] = (lhs[j] + rhs[j]) % kMod;
    }
  }
  return down[1];
}
Poly Log(Poly f) {
  int n = f.size();
  if (n == 1) return {0};
  auto d = Derivative(f);
  f.resize(n - 1);
  d = Multiply(d, Inverse(f));
  d.resize(n - 1);
  return Integral(d);
}
Poly Exp(Poly f) {
  int n = f.size();
  Poly q(1, 1);
  f[0] += 1;
  for (int s = 1; s < n; s <<= 1) {
    if (f.size() < s + s) f.resize(s + s);
    Poly g(f.begin(), f.begin() + s + s);
    Poly h(q.begin(), q.end());
    h.resize(s + s);
    h = Log(h);
    for (int i = 0; i < s + s; ++i) {
      g[i] = (g[i] + kMod - h[i]) % kMod;
    }
    g = Multiply(g, q);
    g.resize(s + s);
    q = g;
  }
  assert(q.size() >= n);
  q.resize(n);
  return q;
}
Poly SquareRootImpl(Poly f) {
  if (f.empty()) return {0};
  int z = QuadraticResidue(f[0], kMod), n = f.size();
  constexpr int kInv2 = (kMod + 1) >> 1;
  if (z == -1) return {-1};
  vector<int> q(1, z);
  for (int s = 1; s < n; s <<= 1) {
    if (f.size() < s + s) f.resize(s + s);
    vector<int> fq(q.begin(), q.end());
    fq.resize(s + s);
    vector<int> f2 = Multiply(fq, fq);
    f2.resize(s + s);
    for (int i = 0; i < s + s; ++i) {
      f2[i] = (f2[i] + kMod - f[i]) % kMod;
    }
    f2 = Multiply(f2, Inverse(fq));
    f2.resize(s + s);
    for (int i = 0; i < s + s; ++i) {
      fq[i] = (fq[i] + kMod - 1LL * f2[i] * kInv2 % kMod) %
     kMod;
    }
    q = fq;
  }
  q.resize(n);
  return q;
}
Poly SquareRoot(Poly f) {
  int n = f.size(), m = 0;
  while (m < n && f[m] == 0) m++;
  if (m == n) return vector<int>(n);
  if (m & 1) return {-1};
  auto s = SquareRootImpl(vector<int>(f.begin() + m, f.end()));
  if (s[0] == -1) return {-1};
  vector<int> res(n);
  for (int i = 0; i < s.size(); ++i) res[i + m / 2] = s[i];
  return res;
}
```

6.5 Newton's Method

Given $F(x)$ where

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

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

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

6.6 General Purpose Numbers

6.6.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(x) = \frac{x}{e^x - 1}$.

6.6.2 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

$$S(n, 1) = S(n, n) = 1$$

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

6.7 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
   - $f(A) = (f(A_0) + f(A_1), f(A_1))$
   - $f^{-1}(A) = (f^{-1}(A_0) - f^{-1}(A_1), f^{-1}(A_1))$

6.8 Simplex Algorithm

Description: maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge 0$. 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;
}
```

6.9 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;
}
```

6.9.1 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 $x_i - x'_i$

6.10 Schreier–Sims Algorithm

```
namespace schreier {
int n;
vector<vector<vector<int>>> bkts, binv;
vector<vector<int>> lk;
vector<int> operator*(const vector<int> &a, const vector<int> &
     b) {
  vector<int> res(a.size());
  for (int i = 0; i < (int)a.size(); ++i) res[i] = b[a[i]];
  return res;
}
vector<int> inv(const vector<int> &a) {
  vector<int> res(a.size());
  for (int i = 0; i < (int)a.size(); ++i) res[a[i]] = i;
  return res;
}
int filter(const vector<int> &g, bool add = true) {
  n = (int)bkts.size();
  vector<int> p = g;
  for (int i = 0; i < n; ++i) {
    assert(p[i] >= 0 && p[i] < (int)lk[i].size());
    int res = lk[i][p[i]];
    if (res == -1) {
      if (add) {
        bkts[i].push_back(p);
        binv[i].push_back(inv(p));
        lk[i][p[i]] = (int)bkts[i].size() - 1;
      }
      return i;
    }
    p = p * binv[i][res];
  }
  return -1;
}
bool inside(const vector<int> &g) { return filter(g, false) ==
     -1; }
void solve(const vector<vector<int>> &gen, int _n) {
  n = _n;
  bkts.clear(), bkts.resize(n);
  binv.clear(), binv.resize(n);
  lk.clear(), lk.resize(n);
  vector<int> iden(n);
  iota(iden.begin(), iden.end(), 0);
```

```
  for (int i = 0; i < n; ++i) {
    lk[i].resize(n, -1);
    bkts[i].push_back(iden);
    binv[i].push_back(iden);
    lk[i][i] = 0;
  }
  for (int i = 0; i < (int)gen.size(); ++i) filter(gen[i]);
  queue<pair<pair<int, int>, pair<int, int>>> upd;
  for (int i = 0; i < n; ++i) {
    for (int j = i; j < n; ++j) {
      for (int k = 0; k < (int)bkts[i].size(); ++k) {
        for (int l = 0; l < (int)bkts[j].size(); ++l)
          upd.emplace(make_pair(i, k), make_pair(j, l));
      }
    }
  }
  while (!upd.empty()) {
    auto a = upd.front().first;
    auto b = upd.front().second;
    upd.pop();
    int res = filter(bkts[a.first][a.second] * bkts[b.first][b.
     second]);
    if (res == -1) continue;
    pair<int, int> pr = make_pair(res, (int)bkts[res].size() -
     1);
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < (int)bkts[i].size(); ++j) {
        if (i <= res) upd.emplace(make_pair(i, j), pr);
        if (res <= i) upd.emplace(pr, make_pair(i, j));
      }
    }
  }
}
long long size() {
  long long res = 1;
  for (int i = 0; i < n; ++i) res = res * bkts[i].size();
  return res;
}}
```

6.11 Berlekamp-Massey Algorithm

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

6.12 Fast Linear Recurrence

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

6.13 Miller Rabin

```
// n < 4759123141    chk = [2, 7, 61]
// n < 1122004669633 chk = [2, 13, 23, 1662803]
// n < 2^64       chk = [2, 325, 9375, 28178, 450775, 9780504,
   1795265022]
vector<long long> chk =
   {2,325,9375,28178,450775,9780504,1795265022};
bool Check(long long a, long long u, long long n, int t) {
  a = fpow(a, u, n);
  if (a == 0 || a == 1 || a == n - 1) return true;
  for (int i = 0; i < t; ++i) {
    a = fmul(a, a, n);
    if (a == 1) return false;
    if (a == n - 1) return true;
  }
  return false;
}
bool IsPrime(long long n) {
  if (n < 2) return false;
  if (n % 2 == 0) return n == 2;
  long long u = n - 1; int t = 0;
  for (; !(u & 1); u >>= 1, ++t);
  for (long long i : chk) {
    if (!Check(i, u, n, t)) return false;
  }
  return true;
}
```

6.14 Pollard's Rho

```
map<long long, int> cnt;
void PollardRho(long long n) {
  if (n == 1) return;
  if (prime(n)) return ++cnt[n], void();
  if (n % 2 == 0) return PollardRho(n / 2), ++cnt[2], void();
  long long x = 2, y = 2, d = 1, p = 1;
  auto f = [&](auto x, auto n, int p) { return (fmul(x, x, n) +
     p) % n; }
  while (true) {
    if (d != n && d != 1) {
      PollardRho(n / d);
      PollardRho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p); y = f(f(y, n, p), n, p);
    d = __gcd(abs(x - y), n);
  }
}
```

6.15 Meissel–Lehmer Algorithm

```
int64_t PrimeCount(int64_t n) {
  if (n <= 1) return 0;
  const int v = sqrt(n);
  vector<int> smalls(v + 1);
  for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
  int s = (v + 1) / 2;
  vector<int> roughs(s);
  for (int i = 0; i < s; ++i) roughs[i] = 2 * i + 1;
  vector<int64_t> larges(s);
  for (int i = 0; i < s; ++i) larges[i] = (n / (2 * i + 1) + 1)
     / 2;
  vector<bool> skip(v + 1);
  int pc = 0;
  for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
      int q = p * p;
      pc++;
      if (1LL * q * q > n) break;
```

```
      skip[p] = 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;
        int64_t d = 1LL * i * p;
        larges[ns] = larges[k] - (d <= v ? larges[smalls[d] -
    pc] : smalls[n / d]) + pc;
        roughs[ns++] = i;
      }
      s = ns;
      for (int j = v / p; j >= p; --j) {
        int c = smalls[j] - pc;
        for (int i = j * p, e = min(i + p, v + 1); i < e; ++i)
    smalls[i] -= c;
      }
    }
  }
  for (int k = 1; k < s; ++k) {
    const int64_t m = n / roughs[k];
    int64_t s = larges[k] - (pc + k - 1);
    for (int l = 1; l < k; ++l) {
      int p = roughs[l];
      if (1LL * p * p > m) break;
      s -= smalls[m / p] - (pc + l - 1);
    }
    larges[0] -= s;
  }
  return larges[0];
}
```

6.16 Discrete Logarithm

Description: to find x such that $x^a \equiv b \pmod p$, let g be the primitive root of p, find k such that $g^k \equiv b \pmod p$ and x can be found by g^d where $ad \equiv k \pmod{p-1}$.

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

6.17 Quadratic Residue

```
int Jacobi(int a, int m) {
  int s = 1;
  for (; m > 1; ) {
    a %= m;
    if (a == 0) return 0;
    const int r = __builtin_ctz(a);
    if ((r & 1) && ((m + 2) & 4)) s = -s;
    a >>= r;
    if (a & m & 2) s = -s;
    swap(a, m);
  }
  return s;
}
int QuadraticResidue(int a, int p) {
  if (p == 2) return a & 1;
  const int jc = Jacobi(a, p);
  if (jc == 0 || jc == -1) return jc;
  int b, d;
  for (; ; ) {
    b = rand() % p;
    d = (1LL * b * b + p - a) % p;
    if (Jacobi(d, p) == -1) break;
  }
  int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
  for (int e = (p + 1) >> 1; e; e >>= 1) {
    if (e & 1) {
      tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p
    ;
      g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
      g0 = tmp;
    }
    tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
    f1 = (2LL * f0 * f1) % p;
    f0 = tmp;
  }
  return g0;
}
```

6.18 Gaussian Elimination

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

6.19 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];
}
```

6.20 μ function

```
int mu[kC], dv[kC];
vector<int> prime;
void Sieve() {
  mu[1] = dv[1] = 1;
  for (int i = 2; i < kC; ++i) {
    if (!dv[i]) {
      dv[i] = i, mu[i] = -1;
      prime.push_back(i);
    }
    for (int j = 0; i * prime[j] < kC; ++j) {
      dv[i * prime[j]] = prime[j];
      mu[i * prime[j]] = -mu[i];
      if (i % prime[j] == 0) {
        mu[i * prime[j]] = 0;
        break;
      }
    }
  }
}
```

6.21 Partition Function

```
void Build(int n) {
  // P[i] = the number of ways to represent i as the sum of a
     non-decreasing sequence.
  vector<pair<int, int>> g = {{0, 0}};
  for (int i = 1; g.back().second <= n; ++i) {
    g.emplace_back(i % 2 ? 1 : kMod - 1, i * (3 * i - 1) / 2);
    g.emplace_back(i % 2 ? 1 : kMod - 1, i * (3 * i + 1) / 2);
  }
  P[0] = P[1] = 1;
  for (int i = 2; i <= n; ++i) {
    for (auto it : g) {
      if (i < it.second) continue;
      (P[i] += 1LL * P[i - it.second] * it.first % kMod) %=
      kMod;
    }
  }
}
```

6.22 $\lfloor \frac{n}{i} \rfloor$ Enumeration

$T_0 = 1, T_i = \lfloor \frac{n}{\lfloor \frac{n}{T_{i-1}+1} \rfloor} \rfloor$

6.23 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;
}
```

6.24 Extended GCD

```
template <typename T> tuple<T, T, T> extgcd(T a, T b) {
  if (!b) return make_tuple(a, 1, 0);
  T d, x, y;
  tie(d, x, y) = extgcd(b, a % b);
  return make_tuple(d, y, x - (a / b) * y);
}
```

6.25 Euclidean Algorithms

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

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

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

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

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

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

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

6.26 Chinese Remainder Theorem

```
long long crt(vector<int> mod, vector<int> a) {
  long long mult = mod[0];
  int n = (int)mod.size();
  long long res = a[0];
  for (int i = 1; i < n; ++i) {
    long long d, x, y;
    tie(d, x, y) = extgcd(mult, mod[i] * 1ll);
    if ((a[i] - res) % d) return -1;
    long long new_mult = mult / __gcd(mult, 1ll * mod[i]) * mod
    [i];
    res += x * ((a[i] - res) / d) % new_mult * mult % new_mult;
    mult = new_mult;
    ((res %= mult) += mult) %= mult;
  }
  return res;
}
```

6.27 Theorem

6.27.1 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 $|\det(\tilde{L}_{rr})|$.

6.27.2 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.

6.27.3 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 $T_{n,k} = kn^{n-k-1}$.

6.27.4 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 $1 \le k \le n$.

6.28 Primes

97, 101, 131, 487, 593, 877, 1087, 1187, 1487, 1787, 3187, 12721, 13331, 14341, 75577, 123457, 222557, 556679, 999983, 1097774749, 1076767633, 100102021, 999997771, 1001010013, 1000512343, 987654361, 999991231, 999888733, 98789101, 987777733, 999991921, 1000000007, 1000000087, 1000000123, 1010101333, 1010102101, 1000000000039, 1000000000000037, 2305843009213693951, 4611686018427387847, 9223372036854775783, 18446744073709551557

7 Dynamic Programming

7.1 Dynamic Convex Hull

```
struct Line {
  mutable int64_t a, b, p;
  bool operator<(const Line &rhs) const { return a < rhs.a; }
  bool operator<(int64_t x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
  static const int64_t kInf = 1e18;
  int64_t Div(int64_t a, int64_t b) { return a / b - ((a ^ b) <
     0 && a % b); }
  bool Isect(iterator x, iterator y) {
    if (y == end()) { x->p = kInf; return false; }
    if (x->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
    else x->p = Div(y->b - x->b, x->a - y->a);
    return x->p >= y->p;
  }
  void Insert(int64_t a, int64_t b) {
    auto z = insert({a, b, 0}), y = z++, x = y;
    while (Isect(y, z)) z = erase(z);
    if (x != begin() && Isect(--x, y)) Isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) Isect(x,
     erase(y));
  }
  int64_t Query(int64_t x) {
    auto l = *lower_bound(x);
    return l.a * 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 Geometry

8.1 Basic

```
bool same(double a, double b) { return abs(a - b) < eps; }

struct P {
  double x, y;
  P() : x(0), y(0) {}
  P(double x, double y) : x(x), y(y) {}
  P operator + (P b) { return P(x + b.x, y + b.y); }
  P operator - (P b) { return P(x - b.x, y - b.y); }
  P operator * (double b) { return P(x * b, y * b); }
  P operator / (double b) { return P(x / b, y / b); }
  double operator * (P b) { return x * b.x + y * b.y; }
  double operator ^ (P b) { return x * b.y - y * b.x; }
  double abs() { return hypot(x, y); }
  P unit() { return *this / abs(); }
  P spin(double o) {
    double c = cos(o), s = sin(o);
    return P(c * x - s * y, s * x + c * y);
  }
  double angle() { return atan2(y, x); }
};

struct L {
  // ax + by + c = 0
  double a, b, c, o;
  P pa, pb;
  L() : a(0), b(0), c(0), o(0), pa(), pb() {}
  L(P pa, P pb) : a(pa.y - pb.y), b(pb.x - pa.x), c(pa ^ pb), o
     (atan2(-a, b)), pa(pa), pb(pb) {}
  P project(P p) { return pa + (pb - pa).unit() * ((pb - pa) *
     (p - pa) / (pb - pa).abs()); }
  P reflect(P p) { return p + (project(p) - p) * 2; }
  double get_ratio(P p) { return (p - pa) * (pb - pa) / ((pb -
     pa).abs() * (pb - pa).abs()); }
};

bool SegmentIntersect(P p1, P p2, P p3, P p4) {
  if (max(p1.x, p2.x) < min(p3.x, p4.x) || max(p3.x, p4.x) <
     min(p1.x, p2.x)) return false;
  if (max(p1.y, p2.y) < min(p3.y, p4.y) || max(p3.y, p4.y) <
     min(p1.y, p2.y)) return false;
  return sign((p3 - p1) ^ (p4 - p1)) * sign((p3 - p2) ^ (p4 -
     p2)) <= 0 &&
      sign((p1 - p3) ^ (p2 - p3)) * sign((p1 - p4) ^ (p2 - p4))
      <= 0;
}

bool parallel(L x, L y) { return same(x.a * y.b, x.b * y.a); }

P Intersect(L x, L y) { return P(-x.b * y.c + x.c * y.b, x.a *
     y.c - x.c * y.a) / (-x.a * y.b + x.b * y.a); }
```

8.2 KD Tree

```
namespace kdt {
int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn], yl[maxn], yr[
     maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
  if (l == r) return -1;
  function<bool(const point &, const point &)> f = [dep](const
     point &a, const point &b) {
    if (dep & 1) return a.x < b.x;
    else return a.y < b.y;
  };
  int m = (l + r) >> 1;
  nth_element(p + l, p + m, p + r, f);
  xl[m] = xr[m] = p[m].x;
  yl[m] = yr[m] = p[m].y;
  lc[m] = build(l, m, dep + 1);
  if (~lc[m]) {
    xl[m] = min(xl[m], xl[lc[m]]);
    xr[m] = max(xr[m], xr[lc[m]]);
    yl[m] = min(yl[m], yl[lc[m]]);
    yr[m] = max(yr[m], yr[lc[m]]);
```

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

8.3 Delaunay Triangulation

Description: Fast Delaunay triangulation assuming no duplicates and not all points collinear (in latter case, result will be empty). Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in ccw order. Each circumcircle will contain none of the input points. If coordinates are ints at most B then T should be large enough to support ints on the order of B^4. We don't need double in Point if the coordinates are integers.

```
namespace delaunay {
// Not equal to any other points.
const Point kA(inf, inf);
bool InCircle(Point p, Point a, Point b, Point c) {
  a = a - p;
  b = b - p;
  c = c - p;
  __int128 x = __int128(a.Norm()) * (b ^ c) + __int128(b.Norm()
    ) * (c ^ a) +
    __int128(c.Norm()) * (a ^ b);
  return x * Sign((b - a) ^ (c - a)) > 0;
}
struct Quad {
  bool mark;
  Quad *o, *rot;
  Point p;
  Quad(Point p) : mark(false), o(nullptr), rot(nullptr), p(p)
     {}
  Point F() { return r()->p; }
  Quad* r() { return rot->rot; }
  Quad* prev() { return rot->o->rot; }
  Quad* next() { return r()->prev(); }
};
Quad* MakeEdge(Point orig, Point dest) {
  Quad* q[4] = {new Quad(orig), new Quad(kA), new Quad(dest),
     new Quad(kA)};
  for (int i = 0; i < 4; ++i) {
    q[i]->o = q[-i & 3];
    q[i]->rot = q[(i + 1) & 3];
  }
  return q[0];
}
void Splice(Quad* a, Quad* b) {
  swap(a->o->rot->o, b->o->rot->o);
  swap(a->o, b->o);
}
Quad* Connect(Quad* a, Quad* b) {
  Quad* q = MakeEdge(a->F(), b->p);
  Splice(q, a->next());
  Splice(q->r(), b);
  return q;
}
pair<Quad*, Quad*> Dfs(const vector<Point>& s, int l, int r) {
  if (r - l <= 3) {
    Quad *a = MakeEdge(s[l], s[l + 1]), *b = MakeEdge(s[l + 1],
      s[r - 1]);
    if (r - l == 2) return {a, a->r()};
    Splice(a->r(), b);
    auto side = (s[l + 1] - s[l]) ^ (s[l + 2] - s[l]);
    Quad* c = side ? Connect(b, a) : nullptr;
    return make_pair(side < 0 ? c->r() : a, side < 0 ? c : b->r
     ());
  }
  int m = (l + r) >> 1;
  auto [ra, a] = Dfs(s, l, m);
  auto [b, rb] = Dfs(s, m, r);
  while (((a->F() - b->p) ^ (a->p - b->p)) < 0 && (a = a->next
     ()) ||
      ((b->F() - a->p) ^ (b->p - a->p)) > 0 && (b = b->r()->o))
     ;
  Quad* base = Connect(b->r(), a);
  auto Valid = [&](Quad* e) {
    return ((base->F() - e->F()) ^ (base->p - e->F())) > 0;
  };
  if (a->p == ra->p) ra = base->r();
  if (b->p == rb->p) rb = base;
  while (true) {
    Quad* lc = base->r()->o;
    if (Valid(lc)) {
      while (InCircle(lc->o->F(), base->F(), base->p, lc->F()))
      {
        Quad* t = lc->o;
        Splice(lc, lc->prev());
        Splice(lc->r(), lc->r()->prev());
        lc = t;
      }
    }
    Quad* rc = base->prev();
    if (Valid(rc)) {
      while (InCircle(rc->prev()->F(), base->F(), base->p, rc->
     F())) {
        Quad* t = rc->prev();
        Splice(rc, rc->prev());
        Splice(rc->r(), rc->r()->prev());
        rc = t;
      }
    }
    if (!Valid(lc) && !Valid(rc)) break;
    if (!Valid(lc) || (Valid(rc) && InCircle(rc->F(), rc->p, lc
     ->F(), lc->p))) {
      base = Connect(rc, base->r());
    } else {
      base = Connect(base->r(), lc->r());
    }
  }
  return make_pair(ra, rb);
}
vector<array<Point, 3>> Triangulate(vector<Point> pts) {
  sort(pts.begin(), pts.end());
  if (pts.size() < 2) return {};
  Quad* e = Dfs(pts, 0, pts.size()).first;
  vector<Quad*> q = {e};
  while (((e->F() - e->o->F()) ^ (e->p - e->o->F())) < 0) e = e
     ->o;
  auto Add = [&]() {
    Quad* c = e;
    do {
      c->mark = true;
      pts.push_back(c->p);
      q.push_back(c->r());
      c = c->next();
    } while (c != e);
  };
  Add();
  pts.clear();
  int ptr = 0;
  while (ptr < q.size()) {
    if (!(e = q[ptr++])->mark) Add();
  }
  vector<array<Point, 3>> res(pts.size() / 3);
  for (int i = 0; i < pts.size(); ++i) res[i / 3][i % 3] = pts[
     i];
  return res;
}
```

```
} // namespace delaunay
```

8.4 Voronoi Diagram

Description: Vertices in Voronoi Diagram are circumcenters of triangles in the Delaunay Triangulation.

```
int gid(P &p) {
  auto it = ptoid.find(p);
  if (it == ptoid.end()) return -1;
  return it->second;
}
L make_line(P p, L l) {
  P d = l.pb - l.pa; d = d.spin(pi / 2);
  P m = (l.pa + l.pb) / 2;
  l = L(m, m + d);
  if (((l.pb - l.pa) ^ (p - l.pa)) < 0) l = L(m + d, m);
  return l;
}
double calc_ans(int i) {
  vector<P> ps = HPI(ls[i]);
  double rt = 0;
  for (int i = 0; i < (int)ps.size(); ++i) {
    rt += (ps[i] ^ ps[(i + 1) % ps.size()]);
  }
  return abs(rt) / 2;
}
void solve() {
  for (int i = 0; i < n; ++i) ops[i] = ps[i], ptoid[ops[i]] = i
     ;
  random_shuffle(ps, ps + n);
  build(n, ps);
  for (auto *t : triang) {
    int z[3] = {gid(t->p[0]), gid(t->p[1]), gid(t->p[2])};
    for (int i = 0; i < 3; ++i) for (int j = 0; j < 3; ++j) if
     (i != j && z[i] != -1 && z[j] != -1) {
      L l(t->p[i], t->p[j]);
      ls[z[i]].push_back(make_line(t->p[i], l));
    }
  }
  vector<P> tb = convex(vector<P>(ps, ps + n));
  for (auto &p : tb) isinf[gid(p)] = true;
  for (int i = 0; i < n; ++i) {
    if (isinf[i]) cout << -1 << '\n';
    else cout << fixed << setprecision(12) << calc_ans(i) << '\
     n';
  }
}
```

8.5 Sector Area

```
// calc area of sector which include a, b
double SectorArea(P a, P b, double r) {
  double o = atan2(a.y, a.x) - atan2(b.y, b.x);
  while (o <= 0) o += 2 * pi;
  while (o >= 2 * pi) o -= 2 * pi;
  o = min(o, 2 * pi - o);
  return r * r * o / 2;
}
```

8.6 Half Plane Intersection

```
bool jizz(L l1,L l2,L l3){
  P p=Intersect(l2,l3);
  return ((l1.pb-l1.pa)^(p-l1.pa))<-eps;
}

bool cmp(const L &a,const L &b){
  return same(a.o,b.o)?(((b.pb-b.pa)^(a.pb-b.pa))>eps):a.o<b.o;
}

// availble area for L l is (l.pb-l.pa)^(p-l.pa)>0
vector<P> HPI(vector<L> &ls){
  sort(ls.begin(),ls.end(),cmp);
  vector<L> pls(1,ls[0]);
  for(int i=0;i<(int)ls.size();++i)if(!same(ls[i].o,pls.back().
     o))pls.push_back(ls[i]);
  deque<int> dq; dq.push_back(0); dq.push_back(1);
#define meow(a,b,c) while(dq.size()>1u && jizz(pls[a],pls[b],
     pls[c]))
  for(int i=2;i<(int)pls.size();++i){
    meow(i,dq.back(),dq[dq.size()-2])dq.pop_back();
    meow(i,dq[0],dq[1])dq.pop_front();
    dq.push_back(i);
  }
  meow(dq.front(),dq.back(),dq[dq.size()-2])dq.pop_back();
  meow(dq.back(),dq[0],dq[1])dq.pop_front();
  if(dq.size()<3u)return vector<P>(); // no solution or
     solution is not a convex
  vector<P> rt;
  for(int i=0;i<(int)dq.size();++i)rt.push_back(Intersect(pls[
     dq[i]],pls[dq[(i+1)%dq.size()]]));
  return rt;
}
```

8.7 Rotating Sweep Line

```
void rotatingSweepLine(vector<pair<int,int>> &ps){
  int n=int(ps.size());
  vector<int> id(n),pos(n);
  vector<pair<int,int>> line(n*(n-1)/2);
  int m=-1;
  for(int i=0;i<n;++i)for(int j=i+1;j<n;++j)line[++m]=make_pair
     (i,j); ++m;
  sort(line.begin(),line.end(),[&](const pair<int,int> &a,const
      pair<int,int> &b)->bool{
    if(ps[a.first].first==ps[a.second].first)return 0;
    if(ps[b.first].first==ps[b.second].first)return 1;
    return (double)(ps[a.first].second-ps[a.second].second)/(ps
     [a.first].first-ps[a.second].first) < (double)(ps[b.first
     ].second-ps[b.second].second)/(ps[b.first].first-ps[b.
     second].first);
  });
  for(int i=0;i<n;++i)id[i]=i;
  sort(id.begin(),id.end(),[&](const int &a,const int &b){
     return ps[a]<ps[b]; });
  for(int i=0;i<n;++i)pos[id[i]]=i;

  for(int i=0;i<m;++i){
    auto l=line[i];
    // meow
    tie(pos[l.first],pos[l.second],id[pos[l.first]],id[pos[l.
     second]])=make_tuple(pos[l.second],pos[l.first],l.second,l
     .first);
  }
}
```

8.8 Triangle Center

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

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

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

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

8.9 Polygon Center

```
Point BaryCenter(vector<Point> &p, int n) {
  Point res(0, 0);
  double s = 0.0, t;
  for (int i = 1; i < p.size() - 1; i++) {
    t = Cross(p[i] - p[0], p[i + 1] - p[0]) / 2;
    s += t;
    res.x += (p[0].x + p[i].x + p[i + 1].x) * t;
    res.y += (p[0].y + p[i].y + p[i + 1].y) * t;
  }
  res.x /= (3 * s);
```

```
  res.y /= (3 * s);
  return res;
}
```

8.10 Maximum Triangle

```
double ConvexHullMaxTriangleArea(Point p[], int res[], int chnum) {
  double area = 0, tmp;
  res[chnum] = res[0];
  for (int i = 0, j = 1, k = 2; i < chnum; i++) {
    while (fabs(Cross(p[res[j]] - p[res[i]], p[res[(k + 1) % chnum]] - p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]))) k = (k + 1) % chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]));
    if (tmp > area) area = tmp;
    while (fabs(Cross(p[res[(j + 1) % chnum]] - p[res[i]], p[res[k]] - p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]))) j = (j + 1) % chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i]]));
    if (tmp > area) area = tmp;
  }
  return area / 2;
}
```

8.11 Point in Polygon

```
int pip(vector<P> ps, P p) {
  int c = 0;
  for (int i = 0; i < ps.size(); ++i) {
    int a = i, b = (i + 1) % ps.size();
    L l(ps[a], ps[b]);
    P q = l.project(p);
    if ((p - q).abs() < eps && l.inside(q)) return 1;
    if (same(ps[a].y, ps[b].y) && same(ps[a].y, p.y)) continue;
    if (ps[a].y > ps[b].y) swap(a, b);
    if (ps[a].y <= p.y && p.y < ps[b].y && p.x <= ps[a].x + (ps[b].x - ps[a].x) / (ps[b].y - ps[a].y) * (p.y - ps[a].y)) ++c;
  }
  return (c & 1) * 2;
}
```

8.12 Circle

```
struct C {
  P c;
  double r;
  C(P c = P(0, 0), double r = 0) : c(c), r(r) {}
};

vector<P> Intersect(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  vector<P> p;
  if (same(a.r + b.r, d)) p.push_back(a.c + (b.c - a.c).unit() * a.r);
  else if (a.r + b.r > d && d + a.r >= b.r) {
    double o = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d));
    P i = (b.c - a.c).unit();
    p.push_back(a.c + i.spin(o) * a.r);
    p.push_back(a.c + i.spin(-o) * a.r);
  }
  return p;
}
double IntersectArea(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  if (d >= a.r + b.r - eps) return 0;
  if (d + a.r <= b.r + eps) return sq(a.r) * acos(-1);
  double p = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d));
  double q = acos((sq(b.r) + sq(d) - sq(a.r)) / (2 * b.r * d));
  return p * sq(a.r) + q * sq(b.r) - a.r * d * sin(p);
}
// remove second level if to get points for line (defalut: segment)
vector<P> CircleCrossLine(P a, P b, P o, double r) {
  double x = b.x - a.x, y = b.y - a.y, A = sq(x) + sq(y), B = 2 * x * (a.x - o.x) + 2 * y * (a.y - o.y);
  double C = sq(a.x - o.x) + sq(a.y - o.y) - sq(r), d = B * B - 4 * A * C;
  vector<P> t;
  if (d >= -eps) {
    d = max(0., d);
    double i = (-B - sqrt(d)) / (2 * A);
    double j = (-B + sqrt(d)) / (2 * A);
    if (i - 1.0 <= eps && i >= -eps) t.emplace_back(a.x + i * x, a.y + i * y);
    if (j - 1.0 <= eps && j >= -eps) t.emplace_back(a.x + j * x, a.y + j * y);
  }
  return t;
}
// calc area intersect by circle with radius r and triangle OAB
double AreaOfCircleTriangle(P a, P b, double r) {
  bool ina = a.abs() < r, inb = b.abs() < r;
  auto p = CircleCrossLine(a, b, P(0, 0), r);
  if (ina) {
    if (inb) return abs(a ^ b) / 2;
    return SectorArea(b, p[0], r) + abs(a ^ p[0]) / 2;
  }
  if (inb) return SectorArea(p[0], a, r) + abs(p[0] ^ b) / 2;
  if (p.size() == 2u) return SectorArea(a, p[0], r) + SectorArea(p[1], b, r) + abs(p[0] ^ p[1]) / 2;
  else return SectorArea(a, b, r);
}
// for any triangle
double AreaOfCircleTriangle(vector<P> ps, double r) {
  double ans = 0;
  for (int i = 0; i < 3; ++i) {
    int j = (i + 1) % 3;
    double o = atan2(ps[i].y, ps[i].x) - atan2(ps[j].y, ps[j].x);
    if (o >= pi) o = o - 2 * pi;
    if (o <= -pi) o = o + 2 * pi;
    ans += AreaOfCircleTriangle(ps[i], ps[j], r) * (o >= 0 ? 1 : -1);
  }
  return abs(ans);
}
```

8.13 Tangent of Circles and Points to Circle

```
vector<L> tangent(C a, C b) {
#define Pij \
  P i = (b.c - a.c).unit() * a.r, j = P(i.y, -i.x);\
  z.emplace_back(a.c + i, a.c + i + j);
#define deo(I,J) \
  double d = (a.c - b.c).abs(), e = a.r I b.r, o = acos(e / d);\
  P i = (b.c - a.c).unit(), j = i.spin(o), k = i.spin(-o);\
  z.emplace_back(a.c + j * a.r, b.c J j * b.r);\
  z.emplace_back(a.c + k * a.r, b.c J k * b.r);
  if (a.r < b.r) swap(a, b);
  vector<L> z;
  if ((a.c - b.c).abs() + b.r < a.r) return z;
  else if (same((a.c - b.c).abs() + b.r, a.r)) { Pij; }
  else {
    deo(-,+);
    if (same(d, a.r + b.r)) { Pij; }
    else if (d > a.r + b.r) { deo(+,-); }
  }
  return z;
}

vector<L> tangent(C c, P p) {
  vector<L> z;
  double d = (p - c.c).abs();
  if (same(d, c.r)) {
    P i = (p - c.c).spin(pi / 2);
    z.emplace_back(p, p + i);
  } else if (d > c.r) {
    double o = acos(c.r / d);
    P i = (p - c.c).unit(), j = i.spin(o) * c.r, k = i.spin(-o) * c.r;
    z.emplace_back(c.c + j, p);
    z.emplace_back(c.c + k, p);
  }
  return z;
}
```

8.14 Area of Union of Circles

```
vector<pair<double, double>> CoverSegment(C &a, C &b) {
  double d = (a.c - b.c).abs();
  vector<pair<double, double>> res;
  if (same(a.r + b.r, d)) ;
  else if (d <= abs(a.r - b.r) + eps) {
    if (a.r < b.r) res.emplace_back(0, 2 * pi);
  } else if (d < abs(a.r + b.r) - eps) {
```

```
    double o = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d)
     ), z = (b.c - a.c).angle();
    if (z < 0) z += 2 * pi;
    double l = z - o, r = z + o;
    if (l < 0) l += 2 * pi;
    if (r > 2 * pi) r -= 2 * pi;
    if (l > r) res.emplace_back(l, 2 * pi), res.emplace_back(0,
      r);
    else res.emplace_back(l, r);
  }
  return res;
}
double CircleUnionArea(vector<C> c) { // circle should be
     identical
  int n = c.size();
  double a = 0, w;
  for (int i = 0; w = 0, i < n; ++i) {
    vector<pair<double, double>> s = {{2 * pi, 9}}, z;
    for (int j = 0; j < n; ++j) if (i != j) {
      z = CoverSegment(c[i], c[j]);
      for (auto &e : z) s.push_back(e);
    }
    sort(s.begin(), s.end());
    auto F = [&] (double t) { return c[i].r * (c[i].r * t + c[i
     ].c.x * sin(t) - c[i].c.y * cos(t)); };
    for (auto &e : s) {
      if (e.first > w) a += F(e.first) - F(w);
      w = max(w, e.second);
    }
  }
  return a * 0.5;
}
```

8.15 Minimun Distance of 2 Polygons

```
// p, q is convex
double TwoConvexHullMinDist(Point P[], Point Q[], int n, int m)
     {
  int YMinP = 0, YMaxQ = 0;
  double tmp, ans = 999999999;
  for (i = 0; i < n; ++i) if(P[i].y < P[YMinP].y) YMinP = i;
  for (i = 0; i < m; ++i) if(Q[i].y > Q[YMaxQ].y) YMaxQ = i;
  P[n] = P[0], Q[m] = Q[0];
  for (int i = 0; i < n; ++i) {
    while (tmp = Cross(Q[YMaxQ + 1] - P[YMinP + 1], P[YMinP] -
     P[YMinP + 1]) > Cross(Q[YMaxQ] - P[YMinP + 1], P[YMinP] -
     P[YMinP + 1])) YMaxQ = (YMaxQ + 1) % m;
    if (tmp < 0) ans = min(ans, PointToSegDist(P[YMinP], P[
     YMinP + 1], Q[YMaxQ]));
    else ans = min(ans, TwoSegMinDist(P[YMinP], P[YMinP + 1], Q
     [YMaxQ], Q[YMaxQ + 1]));
    YMinP = (YMinP + 1) % n;
  }
  return ans;
}
```

8.16 2D Convex Hull

```
bool operator < (const P &a, const P &b) { return same(a.x, b.x
     ) ? a.y < b.y : a.x < b.x; }
bool operator > (const P &a, const P &b) { return same(a.x, b.x
     ) ? a.y > b.y : a.x > b.x; }

#define crx(a, b, c) ((b - a) ^ (c - a))

vector<P> convex(vector<P> ps) {
  vector<P> p;
  sort(ps.begin(), ps.end(), [&] (P a, P b) { return same(a.x,
     b.x) ? a.y < b.y : a.x < b.x; });
  for (int i = 0; i < ps.size(); ++i) {
    while (p.size() >= 2 && crx(p[p.size() - 2], ps[i], p[p.
     size() - 1]) >= 0) p.pop_back();
    p.push_back(ps[i]);
  }
  int t = p.size();
  for (int i = (int)ps.size() - 2; i >= 0; --i) {
    while (p.size() > t && crx(p[p.size() - 2], ps[i], p[p.size
     () - 1]) >= 0) p.pop_back();
    p.push_back(ps[i]);
  }
  p.pop_back();
  return p;
}

int sgn(double x) { return same(x, 0) ? 0 : x > 0 ? 1 : -1; }

P isLL(P p1, P p2, P q1, P q2) {
  double a = crx(q1, q2, p1), b = -crx(q1, q2, p2);
  return (p1 * b + p2 * a) / (a + b);
}

struct CH {
  int n;
  vector<P> p, u, d;
  CH() {}
  CH(vector<P> ps) : p(ps) {
    n = ps.size();
    rotate(p.begin(), min_element(p.begin(), p.end()), p.end())
     ;
    auto t = max_element(p.begin(), p.end());
    d = vector<P>(p.begin(), next(t));
    u = vector<P>(t, p.end()); u.push_back(p[0]);
  }
  int find(vector<P> &v, P d) {
    int l = 0, r = v.size();
    while (l + 5 < r) {
      int L = (l * 2 + r) / 3, R = (l + r * 2) / 3;
      if (v[L] * d > v[R] * d) r = R;
      else l = L;
    }
    int x = l;
    for (int i = l + 1; i < r; ++i) if (v[i] * d > v[x] * d) x
     = i;
    return x;
  }
  int findFarest(P v) {
    if (v.y > 0 || v.y == 0 && v.x > 0) return ((int)d.size() -
      1 + find(u, v)) % p.size();
    return find(d, v);
  }
  P get(int l, int r, P a, P b) {
    int s = sgn(crx(a, b, p[l % n]));
    while (l + 1 < r) {
      int m = (l + r) >> 1;
      if (sgn(crx(a, b, p[m % n])) == s) l = m;
      else r = m;
    }
    return isLL(a, b, p[l % n], p[(l + 1) % n]);
  }
  vector<P> getIS(P a, P b) {
    int X = findFarest((b - a).spin(pi / 2));
    int Y = findFarest((a - b).spin(pi / 2));
    if (X > Y) swap(X, Y);
    if (sgn(crx(a, b, p[X])) * sgn(crx(a, b, p[Y])) < 0) return
     {get(X, Y, a, b), get(Y, X + n, a, b)};
    return {};
  }
  void update_tangent(P q, int i, int &a, int &b) {
    if (sgn(crx(q, p[a], p[i])) > 0) a = i;
    if (sgn(crx(q, p[b], p[i])) < 0) b = i;
  }
  void bs(int l, int r, P q, int &a, int &b) {
    if (l == r) return;
    update_tangent(q, l % n, a, b);
    int s = sgn(crx(q, p[l % n], p[(l + 1) % n]));
    while (l + 1 < r) {
      int m = (l + r) >> 1;
      if (sgn(crx(q, p[m % n], p[(m + 1) % n])) == s) l = m;
      else r = m;
    }
    update_tangent(q, r % n, a, b);
  }
  bool contain(P p) {
    if (p.x < d[0].x || p.x > d.back().x) return 0;
    auto it = lower_bound(d.begin(), d.end(), P(p.x, -1e12));
    if (it->x == p.x) {
      if (it->y > p.y) return 0;
    } else if (crx(*prev(it), *it, p) < -eps) return 0;
    it = lower_bound(u.begin(), u.end(), P(p.x, 1e12), greater<
     P>());
    if (it->x == p.x) {
      if (it->y < p.y) return 0;
    } else if (crx(*prev(it), *it, p) < -eps) return 0;
    return 1;
  }
  bool get_tangent(P p, int &a, int &b) { // b -> a
    if (contain(p)) return 0;
    a = b = 0;
    int i = lower_bound(d.begin(), d.end(), p) - d.begin();
    bs(0, i, p, a, b);
    bs(i, d.size(), p, a, b);
    i = lower_bound(u.begin(), u.end(), p, greater<P>()) - u.
     begin();
    bs((int)d.size() - 1, (int)d.size() - 1 + i, p, a, b);
```

```
    bs((int)d.size() - 1 + i, (int)d.size() - 1 + u.size(), p,
     a, b);
    return 1;
  }
};
```

8.17 3D Convex Hull

```

double absvol(const P a,const P b,const P c,const P d) {
  return abs(((b-a)^(c-a))*(d-a))/6;
}
struct convex3D {
static const int maxn=1010;
struct T{
  int a,b,c;
  bool res;
  T(){}
  T(int a,int b,int c,bool res=1):a(a),b(b),c(c),res(res){}
};
int n,m;
P p[maxn];
T f[maxn*8];
int id[maxn][maxn];
bool on(T &t,P &q){
  return ((p[t.c]-p[t.b])^(p[t.a]-p[t.b]))*(q-p[t.a])>eps;
}
void meow(int q,int a,int b){
  int g=id[a][b];
  if(f[g].res){
    if(on(f[g],p[q]))dfs(q,g);
    else{
      id[q][b]=id[a][q]=id[b][a]=m;
      f[m++]=T(b,a,q,1);
    }
  }
}
void dfs(int p,int i){
  f[i].res=0;
  meow(p,f[i].b,f[i].a);
  meow(p,f[i].c,f[i].b);
  meow(p,f[i].a,f[i].c);
}
void operator()(){
  if(n<4)return;
  if([&](){
    for(int i=1;i<n;++i)if(abs(p[0]-p[i])>eps)return swap(p[1],
     p[i]),0;
    return 1;
  }() || [&](){
    for(int i=2;i<n;++i)if(abs((p[0]-p[i])^(p[1]-p[i]))>eps)
     return swap(p[2],p[i]),0;
    return 1;
  }() || [&](){
    for(int i=3;i<n;++i)if(abs(((p[1]-p[0])^(p[2]-p[0]))*(p[i]-
     p[0]))>eps)return swap(p[3],p[i]),0;
    return 1;
  }())return;
  for(int i=0;i<4;++i){
    T t((i+1)%4,(i+2)%4,(i+3)%4,1);
    if(on(t,p[i]))swap(t.b,t.c);
    id[t.a][t.b]=id[t.b][t.c]=id[t.c][t.a]=m;
    f[m++]=t;
  }
  for(int i=4;i<n;++i)for(int j=0;j<m;++j)if(f[j].res && on(f[j
    ],p[i])){
    dfs(i,j);
    break;
  }
  int mm=m; m=0;
  for(int i=0;i<mm;++i)if(f[i].res)f[m++]=f[i];
}
bool same(int i,int j){
  return !(absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].a])>eps
     || absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].b])>eps ||
     absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].c])>eps);
}
int faces(){
  int r=0;
  for(int i=0;i<m;++i){
    int iden=1;
    for(int j=0;j<i;++j)if(same(i,j))iden=0;
    r+=iden;
  }
  return r;
}
} tb;
```

8.18 Minimum Enclosing Circle

```
pt center(const pt &a, const pt &b, const pt &c) {
  pt p0 = b - a, p1 = c - a;
  double c1 = norm2(p0) * 0.5, c2 = norm2(p1) * 0.5;
  double d = p0 ^ p1;
  double x = a.x + (c1 * p1.y - c2 * p0.y) / d;
  double y = a.y + (c2 * p0.x - c1 * p1.x) / d;
  return pt(x, y);
}

circle min_enclosing(vector<pt> &p) {
  random_shuffle(p.begin(), p.end());
  double r = 0.0;
  pt cent;
  for (int i = 0; i < p.size(); ++i) {
    if (norm2(cent - p[i]) <= r) continue;
    cent = p[i];
    r = 0.0;
    for (int j = 0; j < i; ++j) {
      if (norm2(cent - p[j]) <= r) continue;
      cent = (p[i] + p[j]) / 2;
      r = norm2(p[j] - cent);
      for (int k = 0; k < j; ++k) {
        if (norm2(cent - p[k]) <= r) continue;
        cent = center(p[i], p[j], p[k]);
        r = norm2(p[k] - cent);
      }
    }
  }
  return circle(cent, sqrt(r));
}
```

8.19 Closest Pair

```
double closest_pair(int l, int r) {
  // p should be sorted increasingly according to the x-
     coordinates.
  if (l == r) return 1e9;
  if (r - l == 1) return dist(p[l], p[r]);
  int m = (l + r) >> 1;
  double d = min(closest_pair(l, m), closest_pair(m + 1, r));
  vector<int> vec;
  for (int i = m; i >= l && fabs(p[m].x - p[i].x) < d; --i) vec
     .push_back(i);
  for (int i = m + 1; i <= r && fabs(p[m].x - p[i].x) < d; ++i)
     vec.push_back(i);
  sort(vec.begin(), vec.end(), [&](int a, int b) { return p[a].
     y < p[b].y; });
  for (int i = 0; i < vec.size(); ++i) {
    for (int j = i + 1; j < vec.size() && fabs(p[vec[j]].y - p[
     vec[i]].y) < d; ++j) {
      d = min(d, dist(p[vec[i]], p[vec[j]]));
    }
  }
  return d;
}
```

9 Miscellaneous

9.1 Bitwise Hack

```
long long next_perm(long long v) {
  long long t = v | (v - 1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1))
     ;
}
void subset(long long s) {
  long long sub = s;
  while (sub) sub = (sub - 1) & s;
}
```

9.2 Hilbert's Curve (faster Mo's algorithm)

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

9.3 Mo's Algorithm 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
  }
}
```

9.4 Java

```
import java.io.*;
import java.util.*;
import java.lang.*;
import java.math.*;

public class filename{
  static Scanner in = new Scanner(System.in);
  public static void main(String[] args) throws Exception {
    Scanner fin = new Scanner(new File("infile"));
    PrintWriter fout = new PrintWriter("outfile", "UTF-8");
    fout.println(fin.nextLine());
    fout.close();
    while (in.hasNext()) {
      String str = in.nextLine(); // getline
      String stu = in.next(); // string
    }
    System.out.println("Case #" + t);
    System.out.printf("%d\n", 7122);
    int[][] d = {{7,1,2,2},{8,7}};
    int g = Integer.parseInt("-123");

    long f = (long)d[0][2];

    List<Integer> l = new ArrayList<>();
    Random rg = new Random();
    for (int i = 9; i >= 0; --i) {
      l.add(Integer.valueOf(rg.nextInt(100) + 1));
      l.add(Integer.valueOf((int)(Math.random() * 100) + 1));
    }
    Collections.sort(l, new Comparator<Integer>() {
      public int compare(Integer a, Integer b) { return a - b;
     }
    });
    for (int i = 0; i < l.size(); ++i)
      System.out.print(l.get(i));

    Set<String> s = new HashSet<String>(); // TreeSet
    s.add("jizz");
    System.out.println(s);
    System.out.println(s.contains("jizz"));

    Map<String, Integer> m = new HashMap<String, Integer>();
    m.put("lol", 7122);
    System.out.println(m);
    for(String key: m.keySet())
      System.out.println(key + " : " + m.get(key));
    System.out.println(m.containsKey("lol"));
    System.out.println(m.containsValue(7122));

    System.out.println(Math.PI);
    System.out.println(Math.acos(-1));
    BigInteger bi = in.nextBigInteger(), bj = new BigInteger("
     -7122"), bk = BigInteger.valueOf(17171);
    int sgn = bi.signum(); // sign(bi)
    bi = bi.subtract(BigInteger.ONE).multiply(bj).divide(bj).
     and(bj).gcd(bj).max(bj).pow(87);
    int meow = bi.compareTo(bj); // -1 0 1
    String stz = "f5abd69150";
    BigInteger b16 = new BigInteger(stz, 16);
    System.out.println(b16.toString(2));
  }
}
```

9.5 Dancing Links

```
namespace dlx {
int lt[maxn], rg[maxn], up[maxn], dn[maxn], cl[maxn], rw[maxn],
     bt[maxn], s[maxn], head, sz, ans;
void init(int c) {
  for (int i = 0; i < c; ++i) {
    up[i] = dn[i] = bt[i] = i;
    lt[i] = i == 0 ? c : i - 1;
    rg[i] = i == c - 1 ? c : i + 1;
    s[i] = 0;
  }
  rg[c] = 0, lt[c] = c - 1;
  up[c] = dn[c] = -1;
  head = c, sz = c + 1;
}
void insert(int r, const vector<int> &col) {
  if (col.empty()) return;
  int f = sz;
  for (int i = 0; i < (int)col.size(); ++i) {
    int c = col[i], v = sz++;
    dn[bt[c]] = v;
    up[v] = bt[c], bt[c] = v;
    rg[v] = (i + 1 == (int)col.size() ? f : v + 1);
    rw[v] = r, cl[v] = c;
    ++s[c];
    if (i > 0) lt[v] = v - 1;
  }
  lt[f] = sz - 1;
}
void remove(int c) {
  lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
  for (int i = dn[c]; i != c; i = dn[i]) {
    for (int j = rg[i]; j != i; j = rg[j])
      up[dn[j]] = up[j], dn[up[j]] = dn[j], --s[cl[j]];
  }
}
void restore(int c) {
  for (int i = up[c]; i != c; i = up[i]) {
    for (int j = lt[i]; j != i; j = lt[j])
      ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
  }
  lt[rg[c]] = c, rg[lt[c]] = c;
}
// Call dlx::make after inserting all rows.
void make(int c) {
  for (int i = 0; i < c; ++i)
    dn[bt[i]] = i, up[i] = bt[i];
}
void dfs(int dep) {
  if (dep >= ans) return;
  if (rg[head] == head) return ans = dep, void();
  if (dn[rg[head]] == rg[head]) return;
  int c = rg[head];
  int w = c;
  for (int x = c; x != head; x = rg[x]) if (s[x] < s[w]) w = x;
  remove(w);
  for (int i = dn[w]; i != w; i = dn[i]) {
    for (int j = rg[i]; j != i; j = rg[j]) remove(cl[j]);
    dfs(dep + 1);
    for (int j = lt[i]; j != i; j = lt[j]) restore(cl[j]);
  }
  restore(w);
}
int solve() {
 ans = 1e9, dfs(0);
 return ans;
}}
```

9.6 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
```

```cpp
// 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]++;
}
```

9.7 Manhattan Distance MST

```cpp
void solve(int n) {
  init();
  vector<int> v(n), ds;
  for (int i = 0; i < n; ++i) {
    v[i] = i;
    ds.push_back(x[i] - y[i]);
  }
  sort(ds.begin(), ds.end());
  ds.resize(unique(ds.begin(), ds.end()) - ds.begin());
  sort(v.begin(), v.end(), [&](int i, int j) { return x[i] == x
    [j] ? y[i] > y[j] : x[i] > x[j]; });
  int j = 0;
  for (int i = 0; i < n; ++i) {
    int p = lower_bound(ds.begin(), ds.end(), x[v[i]] - y[v[i
    ]]) - ds.begin() + 1;
    pair<int, int> q = query(p);
    // query return prefix minimum
    if (~q.second) add_edge(v[i], q.second);
    add(p, make_pair(x[v[i]] + y[v[i]], v[i]));
  }
}

void make_graph() {
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
  for (int i = 0; i < n; ++i) x[i] = -x[i];
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
}
```

9.8 IOI 2016 Aliens Trick

```cpp
long long Alien() {
  long long c = kInf;
  for (int d = 60; d >= 0; --d) {
    // cost can be negative as well, depending on the problem.
    if (c - (1LL << d) < 0) continue;
    long long ck = c - (1LL << d);
    pair<long long, int> r = check(ck);
    if (r.second == k) return r.first - ck * k;
    if (r.second < k) c = ck;
  }
  pair<long long, int> r = check(c);
  return r.first - c * k;
}
```

9.9 Matroid Intersection

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

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

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

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

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