ACM-ICPC Team Reference Document
The Codists

Contents

1 Graphs

1.1 Rerooting

```
int N = 7;
vvi g(N);
void dfs0(int node, int par, vvi &g, vi &dp, vi &size) {
   dp[node] = 0; size[node] = 1;
   for (auto nebr : g[node]) {
      if (par != nebr) {
         dfs0(nebr, node, g, dp, size);
         size[node] += size[nebr];
         dp[node] += size[nebr] + dp[nebr];
      }
   }
}
void reroot(int from, int to, vector<int> &dp, vector<int> &size) {
   dp[from] -= size[to] + dp[to];
   size[from] -= size[to];
   size[to] += size[from];
   dp[to] += size[from] + dp[from];
}
void dfs1(int node, int par, vvi &g, vi &dp, vi &ans, vi &size) {
   ans[node] = dp[node];
   for (auto nebr : g[node]) {
      if (par != nebr) {
         reroot(node, nebr, dp, size);
         dfs1(nebr, node, g, dp, ans, size);
         reroot(nebr, node, dp, size);
      }
   }
}
void edge(int a, int b, vvi &g) {
   a--; b--;
   g[a].push_back(b);
   g[b].push_back(a);
}
vector<int> pathSum(vvi &g, int N) {
   vector<int> dp(N), ans(N), size(N);
   dfs0(0, -1, g, dp, size);
   dfs1(0, -1, g, dp, ans, size);
   return ans;
}
```

1.2 Max Flow

```
struct MfEdge {
 int v, cap;
 int backid; // id to the back edge
};
struct MaxFlow {
 vector<vector<int>> g; // integers represent edges' ids
 vector<MfEdge> edges; // edges.size() should always be even
 int n, s, t; // n = # vertices, s = src vertex, t = sink vertex
 int find_path() {
  const int inf = int(1e9 + 7);
  vector<int> from(n, -1), used_edge(n, -1);

  vector<int> visited(n, -1); queue<int> q;
  q.push(s); visited[s] = true;
  while (!visited[t] && !q.empty()) {
   int u = q.front();
   q.pop();
   for (int eid : g[u]) {
    int v = edges[eid].v;
    if (edges[eid].cap > 0 && !visited[v]) {
     from[v] = u, used_edge[v] = eid;
     q.push(v); visited[v] = true;
     if (v == t) break;
    }
   }
  }
  int f = inf;
  if (from[t] != -1) {
   for (int v = t; from[v] > -1; v = from[v]) {
    f = min(edges[used_edge[v]].cap, f);
   }
   for (int v = t; from[v] > -1; v = from[v]) {
    int backid = edges[used_edge[v]].backid;
    edges[used_edge[v]].cap -= f;
    edges[backid].cap += f;
   }
  }
  return (f == inf ? 0 : f);
 }
 int get() {
  int mf = 0, d;
  while ((d = find_path())) mf += d;
  return mf;
```

```
  }
};
```

1.3 Prim Dense Graphs

```
int n;
vector<vector<int>> adj; // adjacency matrix of graph
const int INF = 1000000000; // weight INF means there is no edge
struct Edge {
   int w = INF, to = -1;
};
void prim() {
   int total_weight = 0;
   vector<bool> selected(n, false);
   vector<Edge> min_e(n);
   min_e[0].w = 0;
   for (int i = 0; i < n; ++i) {
      int v = -1;
      for (int j = 0; j < n; ++j) {
         if (!selected[j] && (v == -1 || min_e[j].w < min_e[v].w))
            v = j;
      }
      if (min_e[v].w == INF) {
         cout << "No MST!" << endl;
         exit(0);
      }
      selected[v] = true;
      total_weight += min_e[v].w;
      if (min_e[v].to != -1)
         cout << v << " " << min_e[v].to << endl;
      for (int to = 0; to < n; ++to) {
         if (adj[v][to] < min_e[to].w)
            min_e[to] = {adj[v][to], v};
      }
   }
   cout << total_weight << endl;
}
```

1.4 Scc

```
int n, m;
vector<vector<int>> adj, adj2, components;
vector<int> nodes, root;
vector<bool> visited;
void dfs(int v) {
   visited[v] = true;
   for (auto u : adj[v]) {
      if (!visited[u]) {
         dfs(u);
      }
   }
   nodes.push_back(v);
}
void dfs2(int v) {
   visited[v] = true;
   components.back().push_back(v);
   root[v] = components.size() - 1;
   for (auto u : adj2[v]) {
      if (!visited[u]) {
         dfs2(u);
      }
   }
}
adj.assign(n, vector<int>()); adj2.assign(n, vector<int>());
visited.assign(n, false); root.assign(n, 0);
for (int i = 0; i < n; i++) {
   if (!visited[i]) { dfs(i); }
}
reverse(all(nodes));
visited.assign(n, false);
for (int i = 0; i < n; i++) {
   if (!visited[nodes[i]]) {
      components.push_back(vi());
      dfs2(nodes[i]);
   }
}
```

1.5 Bridges

```
#define SZ 100
bool M[SZ][SZ];
```

```
int N, colour[SZ], dfsNum[SZ], num, pos[SZ], leastAncestor[SZ],
    parent[SZ];
void dfs(int u) {
  int v;
  stack<int> S;
  S.push(u);
  while (!S.empty()) {
    v = S.top();
    if (colour[v] == 0) {
      colour[v] = 1;
      dfsNum[v] = num++;
      leastAncestor[v] = num;
    }
    for (; pos[v] < N; ++pos[v]) {
      if (M[v][pos[v]] && pos[v] != parent[v]) {
        if (colour[pos[v]] == 0) {
          parent[pos[v]] = v;
          S.push(pos[v]);
          break;
        } else
          leastAncestor[v] < ? = dfsNum[pos[v]];
      }
    }
    if (pos[v] == N) {
      colour[v] = 2;
      S.pop();
      if (v != u) leastAncestor[parent[v]] < ? = leastAncestor[v];
    }
  }
}
void Bridge_detection() {
  memset(colour, 0, sizeof(colour));
  memset(pos, 0, sizeof(pos));
  memset(parent, -1, sizeof(parent));
  num = 0;
  int ans = 0;
  for (int i = 0; i < N; i++)
    if (colour[i] == 0) dfs(i);
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      if (parent[j] == i && leastAncestor[j] > dfsNum[i]) {
        printf("%d - %d\n", i, j);
        ++ans;
      }
  printf("%d bridges\n", ans);
}
```

1.6 Maximum Matching

```
#include <bits/stdc++.h>
using namespace std;
#define NIL 0
#define INF INT_MAX
class BipGraph {
public:
   int m, n;
   list<int> *adj;
   int *pairU, *pairV, *dist;
   BipGraph(int m, int n);
   void addEdge(int u, int v);
   bool bfs();
   bool dfs(int u);
   int hopcroftKarp();
};
int BipGraph::hopcroftKarp() {
   pairU = new int[m + 1];
   pairV = new int[n + 1];
   dist = new int[m + 1];
   for (int u = 0; u <= m; u++)
      pairU[u] = NIL;
   for (int v = 0; v <= n; v++)
      pairV[v] = NIL;
   int result = 0;
   while (bfs()) {
      for (int u = 1; u <= m; u++)
         if (pairU[u] == NIL && dfs(u))
            result++;
   }
   return result;
}
bool BipGraph::bfs() {
   queue<int> Q;
   for (int u = 1; u <= m; u++) {
      if (pairU[u] == NIL) {
         dist[u] = 0;
         Q.push(u);
      } else
```

```
            dist[u] = INF;
      }
      dist[NIL] = INF;
      while (!Q.empty()) {
         int u = Q.front();
         Q.pop();
         if (dist[u] < dist[NIL]) {
            list<int>::iterator i;
            for (i = adj[u].begin(); i != adj[u].end(); ++i) {
               int v = *i;
               if (dist[pairV[v]] == INF) {
                  dist[pairV[v]] = dist[u] + 1;
                  Q.push(pairV[v]);
               }
            }
         }
      }
      return (dist[NIL] != INF);
}
bool BipGraph::dfs(int u) {
      if (u != NIL) {
         list<int>::iterator i;
         for (i = adj[u].begin(); i != adj[u].end(); ++i) {
            int v = *i;
            if (dist[pairV[v]] == dist[u] + 1) {
               if (dfs(pairV[v]) == true) {
                  pairV[v] = u;
                  pairU[u] = v;
                  return true;
               }
            }
         }
         dist[u] = INF;
         return false;
      }
      return true;
}
BipGraph::BipGraph(int m, int n) {
      this->m = m;
      this->n = n;
      adj = new list<int>[m + 1];
}
void BipGraph::addEdge(int u, int v) {
      adj[u].push_back(v);
}
```

```
int main() {
      int n, m, k;
      cin >> n >> m >> k;
      BipGraph g(n, m);
      int a, b;
      for (int i = 0; i < k; i++) {
         cin >> a >> b;
         g.addEdge(a, b);
      }
      cout << g.hopcroftKarp() << endl;
      for (int i = 1; i <= g.m; i++) {
         if (g.pairU[i] != NIL)
            cout << i << " " << g.pairU[i] << endl;
      }
}
```

1.7 Second Mst

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
vector<pair<int, pair<int, int>>> edg;
vector<vector<pair<int, int>>> mst;
vector<int> h, parent, Rank;
vector<vector<int>> up, dp;
void dfs(int u, int p, int log, int d) {
      up[u][0] = p;
      dp[u][0] = d;
      h[u] = 1 + h[p];
      // for (int i = 1; i <= log; i++)
      // {
      // up[u][i] = up[up[u][i - 1]][i - 1];
      // dp[u][i] = max(dp[u][i - 1], dp[up[u][i - 1]][i - 1]);
      // }
      for (auto v : mst[u]) {
         if (v.first != p) {
            dfs(v.first, u, log, v.second);
         }
      }
}
int lca(int u, int v, int log) {
      int ans = -2;
```

```
    if (h[u] < h[v]) {
        swap(u, v);
    }
    for (int i = log; i >= 0; i--) {
        if (h[u] - pow(2, i) >= h[v]) {
            ans = max(ans, dp[u][i]);
            u = up[u][i];
        }
    }
    if (u == v) {
        return ans;
    }
    for (int i = log; i >= 0; i--) {
        if (up[u][i] != up[v][i]) {
            ans = max(ans, max(dp[u][i], dp[v][i]));
            u = up[u][i];
            v = up[v][i];
        }
    }
    ans = max(ans, max(dp[u][0], dp[v][0]));
    return ans;
}
int findParent(int u) {
    if (u == parent[u])
        return u;
    return parent[u] = findParent(parent[u]);
}
void unionByRank(int x, int y) {
    int px = findParent(x);
    int py = findParent(y);
    if (px == py)
        return;
    if (Rank[px] < Rank[py])
        parent[px] = py;
    else if (Rank[px] > Rank[py])
        parent[py] = px;
    else {
        parent[px] = py;
        Rank[py]++;
    }
}
signed main() {
    int n, m;
    cin >> n >> m;
    int a, b, c;
    int log = (int)ceil(log2(n));
    map<pair<int, int>, int> edg_ind;
    parent.assign(n + 1, 0);
    Rank.assign(n + 1, 0);
    h.assign(n + 1, 0);
    up.assign(n + 1, vector<int>(log + 1, 0));
    dp.assign(n + 1, vector<int>(log + 1, 0));
    for (int i = 0; i <= n; i++)
        parent[i] = i;
    mst.assign(n + 1, vector<pair<int, int>>());
    for (int i = 0; i < m; i++) {
        cin >> a >> b >> c;
        edg.push_back({c, {a, b}});
        edg_ind[{a, b}] = i;
    }
    sort(edg.begin(), edg.end());
    vector<int> vis(m, 0);
    vector<int> ans(m, 0);
    int MST = 0;
    for (int i = 0; i < m; i++) {
        if (findParent(edg[i].second.first) !=
            findParent(edg[i].second.second)) {
            unionByRank(edg[i].second.first, edg[i].second.second);
            MST += edg[i].first;
            ans[edg_ind[{edg[i].second.first, edg[i].second.second}]]
                = -1;
            vis[i] = 1;
            mst[edg[i].second.first].push_back({edg[i].second.second,
                edg[i].first});
            mst[edg[i].second.second].push_back({edg[i].second.first,
                edg[i].first});
        }
    }
    int pp = -1;
    for (int i = 1; i <= n; i++) {
        if (parent[i] == i)
            pp = i;
    }
    dfs(pp, pp, log, 0);
    for (int i = 1; i <= log; i++) {
        for (int j = 1; j <= n; ++j) {
            if (up[j][i - 1] != -1) {
                int v = up[j][i - 1];
                up[j][i] = up[v][i - 1];
                dp[j][i] = max(dp[j][i - 1], dp[v][i - 1]);
```

```
            }
        }
    }
    for (int i = 0; i < m; i++) {
        if (!vis[i]) {
            int rem = lca(edg[i].second.first, edg[i].second.second,
                log);
            ans[edg_ind[{edg[i].second.first, edg[i].second.second}]]
                = MST - rem + edg[i].first;
        }
    }
    for (int i = 0; i < m; i++) {
        if (ans[i] == -1)
            cout << MST << endl;
        else
            cout << ans[i] << endl;
    }
}
```

1.8 Max Flow With Min Cost

```
struct McMaxFlow {
  struct MfEdge { int v, cap, cpu, backid; };
  struct FlowResult { int flow, cost; };
  vector<vector<int>> g; // integers represent edges' ids
  vector<MfEdge> edges; // edges.size() should always be even
  int n, s, t; // n = # vertices, s = src vertex, t = sink vertex

  // Directed Edge u - > v with capacity 'cap' and cost per unit
      'cpu'
  void add_edge(int u, int v, int cap, int cpu) {
    int eid = edges.size();
    g[u].push_back(eid);
    g[v].push_back(eid + 1);
    edges.push_back((MfEdge){v, cap, cpu, eid + 1});
    edges.push_back((MfEdge){u, 0, -cpu, eid});
  }

  FlowResult find_path() {
    const int inf = int(1e9 + 7);
    vector<int> from(n, -1), used_edge(n, -1);
    vector<int> dist(n, inf);
    queue<int> q; vector<bool> queued(n, false);
    dist[s] = 0; q.push(s); queued[s] = true;

    while (!q.empty()) {
      const int u = q.front(); q.pop();
      queued[u] = false;

      for (int eid : g[u]) {
        int v = edges[eid].v;
        int cand_dist = dist[u] + edges[eid].cpu;
        if (edges[eid].cap > 0 && cand_dist < dist[v]) {
          dist[v] = cand_dist;
          from[v] = u; used_edge[v] = eid;
          if (!queued[v]) { q.push(v); queued[v] = true; }
        }
      }
    }

    int f = 0, fcost = 0;
    if (from[t] != -1) {
      f = inf;
      for (int v = t; from[v] > -1; v = from[v]) {
        f = min(edges[used_edge[v]].cap, f);
        fcost += edges[used_edge[v]].cpu;
      }
      for (int v = t; from[v] > -1; v = from[v]) {
        int backid = edges[used_edge[v]].backid;
        edges[used_edge[v]].cap -= f;
        edges[backid].cap += f;
      }
      fcost *= f;
    }

    return (FlowResult){f, fcost};
  }

  FlowResult get() {
    FlowResult res = {0, 0};
    while (true) {
      FlowResult fr = find_path();
      if (fr.flow == 0) break;
      res.flow += fr.flow;
      res.cost += fr.cost;
    }
```

```
   return res;
  }
};
```

1.9 Dijkstra

```
vector<vector<pair<int, int>>> adj;
vector<bool> visited;
vector<int> dist;

void dijkstra(int source) {
   priority_queue<pair<int, int>, vector<pair<int, int>>,
       greater<pair<int, int>>> pq;
   dist[source] = 0; pq.push({0, source});
   while (!pq.empty()) {
      int source = pq.top().second;
      pq.pop();
      if (visited[source]) {
         continue;
      }
      visited[source] = true;
         for (auto it : adj[source]) {
            if (dist[it.first] > dist[source] + it.second) {
               dist[it.first] = dist[source] + it.second;
               pq.push({dist[it.first], it.first});
         }
      }
   }
}
```

1.10 Articulation Points

```
#define SZ 100
bool M[SZ][SZ];
int N, colour[SZ], dfsNum[SZ], num, pos[SZ], leastAncestor[SZ],
    parent[SZ];
int dfs(int u) {
  int ans = 0, cont = 0, v;
  stack<int> S;
  S.push(u);
  while (!S.empty()) {
    v = S.top();
    if (colour[v] == 0) {
      colour[v] = 1;
      dfsNum[v] = num++;
      leastAncestor[v] = num;
    }
    for (; pos[v] < N; ++pos[v]) {
      if (M[v][pos[v]] && pos[v] != parent[v]) {
        if (colour[pos[v]] == 0) {
          parent[pos[v]] = v;
          S.push(pos[v]);
          if (v == u) ++cont;
          break;
        } else
          leastAncestor[v] < ? = dfsNum[pos[v]];
      }
    }
    if (pos[v] == N) {
      colour[v] = 2;
      S.pop();
      if (v != u) leastAncestor[parent[v]] < ? = leastAncestor[v];
    }
  }
  if (cont > 1) {
    ++ans;
    printf("%d\n", u);
  }
  for (int i = 0; i < N; ++i) {
    if (i == u) continue;
    for (int j = 0; j < N; j++)
      if (M[i][j] && parent[j] == i && leastAncestor[j] >=
          dfsNum[i]) {
        printf("%d\n", i);
        ++ans;
        break;
      }
  }
  return ans;
}
void Articulation_points() {
  memset(colour, 0, sizeof(colour));
  memset(pos, 0, sizeof(pos));
  memset(parent, -1, sizeof(parent));
```

```
  num = 0;
  int total = 0;
  for (int i = 0; i < N; ++i)
    if (colour[i] == 0) total += dfs(i);
  printf("# Articulation Points : %d\n", total);
}
```

1.11 Kruskal

```
class Edge {
public:
   int u, v, w;
};
int n, m;
vector<Edge> edgeList(m);
DSU dsu(n);
sort(all(edgeList), [](Edge &a, Edge &b) { return a.w < b.w; });
int cost = 0;
for (Edge e : edgeList) {
   if (dsu.find_set(e.u) != dsu.find_set(e.v)) {
      cost += e.w;
      dsu.union_set(e.u, e.v);
   }
}
```

1.12 All Longest Paths

```
void dfs1(int v, int p) {
   f[v] = h[v] = 0;
   for (auto u : adj[v]) {
      if (u == p) continue;
      dfs1(u, v);
      if (f[v] <= f[u] + 1) {
         h[v] = f[v]; f[v] = f[u] + 1;
      } else if (h[v] < f[u] + 1) {
         h[v] = f[u] + 1;
      }
   }
}

void dfs2(int v, int p) {
   for (auto u : adj[v]) {
      if (u == p) continue;
      if (f[v] == f[u] + 1) {
         g[u] = max(g[v] + 1, h[v] + 1);
      } else {
         g[u] = max(g[v] + 1, f[v] + 1);
      }
      dfs2(u, v);
   }
}
```

2 Dynamic Programming

2.1 Grundy Number

```
const int BOARD_LEN = 15;
int nimber(int r, int c) {
      static std::map<std::pair<int, int>, int> cache;
      if (cache.count({r, c})) { return cache[{r, c}]; }
      if (r < 0 || BOARD_LEN <= r || c < 0 || BOARD_LEN <= c) {
            return -1; // return -1 to not interfere with the mex
                operations
      }
      std::set<int> reachable{nimber(r - 2, c + 1), nimber(r - 2,
          c - 1),
                              nimber(r + 1, c - 2), nimber(r - 1, c -
                                  2)};

      int ret = 0;
      while (reachable.count(ret)) { ret++; }
      return cache[{r, c}] = ret;
}

int main() {
      int test_num; cin >> test_num;
      for (int t = 0; t < test_num; t++) {
            int coin_xor = 0, coin_num;
      cin >> coin_num;
            for (int i = 0; i < coin_num; i++) {
```

```
            int r, c; cin >> r >> c;
            coin_xor ^= nimber(r - 1, c - 1);
        }
        cout << (coin_xor == 0 ? "Second" : "First") << '\n';
    }
}
```

2.2 Digit Dp

```
// Counting Numbers (CSES DP)
int a, b, length;
vector<vvvi> dp;
int f(string &c, int index, int previous, int leading_zero, int
    is_lesser) {
   if (index == c.size()) {
      return 1;
   }
   if (dp[index][previous][leading_zero][is_lesser] != -1) {
      return dp[index][previous][leading_zero][is_lesser];
   }
   int answer = 0;
   int limit = 9;
   if (!is_lesser) {
      limit = c[index] - '0';
   }
   for (int i = 0; i <= limit; i++) {
      if (leading_zero == 0 && (i == previous)) {
         continue;
      }
      int new_leading_zero = (leading_zero == 1 && i == 0) ? 1 : 0;
      if (is_lesser) {
         answer += f(c, index + 1, i, new_leading_zero, is_lesser);
      } else {
         answer += f(c, index + 1, i, new_leading_zero, i < limit);
      }
   }
   return dp[index][previous][leading_zero][is_lesser] = answer;
   // return answer;
}
void solve() {
   cin >> a >> b;
   string r = to_string(b);
   string l = to_string(a - 1);
   dp.assign(r.size(), vvvi(10, vvi(2, vi(2, -1))));
   int answer = f(r, 0, 0, 1, 0);
   dp.assign(l.size(), vvvi(10, vvi(2, vi(2, -1))));
   if (a != 0)
      answer -= f(l, 0, 0, 1, 0);
   print(answer);
}
// Digit Sum (Atcoder DP)
string K;
int D;
vvvi dp;
int f(int index, int sum, int is_lesser) {
   if (index == K.size()) {
      return ((sum % D) == 0);
   }
   if (dp[index][sum][is_lesser] != -1) {
      return dp[index][sum][is_lesser];
   }
   int answer = 0;
   int limit = (is_lesser) ? 9 : K[index] - '0';
   for (int i = 0; i <= limit; i++) {
      int new_is_lesser = (is_lesser || (i < limit)) ? 1 : 0;
      answer += f(index + 1, (sum + i) % D, new_is_lesser);
      answer %= MOD;
   }
   return dp[index][sum][is_lesser] = answer % MOD;
}
void solve() {
   cin >> K >> D;
   dp.assign(K.size(), vvi(D, vi(2, -1)));
   int answer = f(0, 0, 0);
   K = "0";
   dp.assign(K.size(), vvi(D, vi(2, -1)));
   answer = (answer - f(0, 0, 0) + MOD) % MOD;
   print(answer);
}
```

2.3 Hamiltonian

```
vi adj[21];
```

```
ll dp[1 << 20][21]; // amount of flights of subset S ending at
    city d
int n, m; cin >> n >> m;
for (int i = 0; i < m; i++) {
   int a, b; cin >> a >> b;
   adj[b].pb(a);
}
dp[1][1] = 1; // there is one way to fly from 1 to itself
for (int s = 2; s < (1 << n); s++) { // we start from the second
    city
   if ((s & (1 << (n - 1))) && s != ((1 << n) - 1)) continue;
   for (int d = 1; d <= n; d++) { // loop through each city
      if ((s & (1 << (d - 1))) == 0) continue;
      for (int v : adj[d]) {
         if (s & (1 << (v - 1))) { // if v is in the mask
            dp[s][d] += dp[s - (1 << (d - 1))][v];
            dp[s][d] %= mod;
         }
      }
   }
}
cout << dp[(1 << n) - 1][n] % mod;
```

2.4 Broken Profile

```
int n, m;
vvi dp;
void getMasks(int i, int mask, int nextMask, vi &nextMasks) {
   if (i == n) {
      nextMasks.push_back(nextMask); return;
   } if (i + 1 < n && ((1 << i) & mask) == 0 && ((1 << (i + 1)) &
       mask) == 0) {
      getMasks(i + 2, mask, nextMask, nextMasks);
   } if (((1 << i) & mask) == 0) {
      getMasks(i + 1, mask, nextMask + (1 << i), nextMasks);
   } if ((((1 << i) & (mask)) != 0)) {
      getMasks(i + 1, mask, nextMask, nextMasks);
   } return;
}

int f(int i, int mask) {
   if (i == m) {
      if (mask) { return 0; }
      return 1;
   }
   if (dp[i][mask] != -1) { return dp[i][mask]; }
   int answer = 0; vi nextMasks;
   getMasks(0, mask, 0, nextMasks);
   for (auto x : nextMasks) {
      answer += f(i + 1, x); answer %= MOD;
   }
   return dp[i][mask] = answer;
}

void solve() {
   cin >> n >> m;
   dp.assign(m, vi(1 << n, -1));
   int answer = f(0, 0);
   print(answer);
}
```

2.5 Bitmask Dp

```
int n;
vector<vector<int>> a;
vector<int> dp(1 << 21, -1);
cin >> n;
a.assign(n, vector<int>(n, 0));
dp.assign(1 << n, 0);
for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
      cin >> a[i][j];
   }
}
// Recursive
int f(int bitmask) {
   if (bitmask == (1 << n) - 1) {
      return 1;
   }
   if (dp[bitmask] != -1)
      return dp[bitmask];
   int answer = 0;
   int males = __builtin_popcount(bitmask);
   for (int i = 0; i < n; i++) {
```

```
      if (a[males][i] == 1 && (bitmask & (1 << i)) == 0) {
         answer += f(bitmask | (1 << i));
         answer %= MOD;
      }
   }
   return dp[bitmask] = answer;
}
// Iterative
dp[0] = 1;
for (int i = 0; i < (1 << n); i++) {
   int males = __builtin_popcount(i);
   for (int j = 0; j < n; j++) {
      if ((i & (1 << j)) || a[males][j] == 0)
         continue;
      dp[i | (1 << j)] += dp[i];
      dp[i | (1 << j)] %= MOD;
   }
}
print(dp[(1 << n) - 1]);
```

2.6 Knapsack

```
vector<int> price(n, 0), pages(n, 0), dp(x + 1, 0);
for (int i = 0; i < n; i++) {
   for (int j = x; j >= price[i]; j--) {
      dp[j] = max(dp[j], dp[j - price[i]] + pages[i]);
   }
}
print(dp[x]);
```

3 Math

3.1 Divisors

```
// Number of divisors
long long numberOfDivisors(long long num) {
   long long total = 1;
   for (int i = 2; (long long)i * i <= num; i++) {
      if (num % i == 0) {
         int e = 0;
         do {
            e++;
            num /= i;
         } while (num % i == 0);
         total *= e + 1;
      }
   }
   if (num > 1) {
      total *= 2;
   }
   return total;
}
// Sum of divisors
long long SumOfDivisors(long long num) {
   long long total = 1;
   for (int i = 2; (long long)i * i <= num; i++) {
      if (num % i == 0) {
         int e = 0;
         do {
            e++;
            num /= i;
         } while (num % i == 0);

         long long sum = 0, pow = 1;
         do {
            sum += pow;
            pow *= i;
         } while (e-- > 0);
         total *= sum;
      }
   }
   if (num > 1) {
      total *= (1 + num);
   }
   return total;
}
```

3.2 Binomial Coefficients

```
vector<int> factorial(MAXN + 1);
vector<int> inv(MAXN + 1);

factorial[0] = 1;
for (int i = 1; i <= MAXN; i++)
   factorial[i] = factorial[i - 1] * i % m;

inv[0] = 0; inv[1] = 1;
for (int a = 2; a <= MAXN; ++a)
   inv[a] = m - (long long)(m / a) * inv[m % a] % m;

long long binomial_coefficient(int n, int k) {
   return factorial[n] * inverse(factorial[k] * factorial[n - k] %
       m) % m;
}
```

3.3 Chinese Remainder Theorem

```
// rem y mod tienen el mismo numero de elementos
long long chinese_remainder(vector<int> rem, vector<int> mod) {
   long long ans = rem[0], m = mod[0];
   int n = rem.size();
   for (int i = 1; i < n; ++i) {
      int a = modular_inverse(m, mod[i]);
      int b = modular_inverse(mod[i], m);
      ans = (ans * b * mod[i] + rem[i] * a * m) % (m * mod[i]);
      m *= mod[i];
   }

   return ans;
}
// cp-algo
struct Congruence {
   long long a, m;
};
long long chinese_remainder_theorem(vector<Congruence> const
    &congruences) {
   long long M = 1;
   for (auto const &congruence : congruences) {
      M *= congruence.m;
   }
   long long solution = 0;
   for (auto const &congruence : congruences) {
      long long a_i = congruence.a;
      long long M_i = M / congruence.m;
      long long N_i = mod_inv(M_i, congruence.m);
      solution = (solution + a_i * M_i % M * N_i) % M;
   }
   return solution;
}
```

3.4 Modular Exponentiation

```
long long binpow(long long a, long long b) {
   long long res = 1;
   while (b > 0) {
      if (b & 1)
         res = res * a;
      a = a * a;
      b >>= 1;
   }
   return res;
}
```

3.5 Extended Gcd

```
// Extended Euclidean
int gcd(int a, int b, int &x, int &y) {
   if (b == 0) {
      x = 1;
      y = 0;
      return a;
   }
   int x1, y1;
   int d = gcd(b, a % b, x1, y1);
   x = y1;
   y = x1 - y1 * (a / b);
   return d;
}
// Inverse Mod using Extended Euclidean
int x, y;
```

```
int g = extended_euclidean(a, m, x, y);
if (g != 1) {
   cout << "No solution!";
} else {
   x = (x % m + m) % m;
   cout << x << endl;
}
```

3.6 Euler Totient

```
// Euler totient function of n in O(sqrt n)
int phi(int n) {
   int result = n;
   for (int i = 2; i * i <= n; i++) {
      if (n % i == 0) {
         while (n % i == 0)
            n /= i;
         result -= result / i;
      }
   }
   if (n > 1)
      result -= result / n;
   return result;
}
// Euler totient function from 1 to n in O(nloglogn)
void phi_1_to_n(int n) {
   vector<int> phi(n + 1);
   for (int i = 0; i <= n; i++)
      phi[i] = i;

   for (int i = 2; i <= n; i++) {
      if (phi[i] == i) {
         for (int j = i; j <= n; j += i)
            phi[j] -= phi[j] / i;
      }
   }
}
```

3.7 Sieve Of Eratosthenes

```
vector<bool> prime(2e7 + 1, true);
prime[0] = prime[1] = false;
int n = 2e6 + 1;
for (int p = 2; p <= n; p++) {
   if (prime[p] == true) {
      for (int i = p * p; i <= n; i += p)
         prime[i] = false;
   }
}
```

3.8 Ntt

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1 << 20;
void fft(vector<int> &a, bool invert) {
   int n = a.size();
   for (int i = 1, j = 0; i < n; i++) {
      int bit = n >> 1;
      for (; j & bit; bit >>= 1)
         j ^= bit;
      j ^= bit;
      if (i < j)
         swap(a[i], a[j]);
   }
   for (int len = 2; len <= n; len <<= 1) {
      int wlen = invert ? root_1 : root;
      for (int i = len; i < root_pw; i <<= 1)
         wlen = (int)(1LL * wlen * wlen % mod);
      for (int i = 0; i < n; i += len) {
         int w = 1;
         for (int j = 0; j < len / 2; j++) {
            int u = a[i + j], v = (int)(1LL * a[i + j + len / 2] *
                w % mod);
            a[i + j] = u + v < mod ? u + v : u + v - mod;
            a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;
            w = (int)(1LL * w * wlen % mod);
         }
      }
   }
   if (invert) {
```

```
      int n_1 = inverse(n, mod);
      for (int &x : a)
         x = (int)(1LL * x * n_1 % mod);
   }
}
```

3.9 Pascal Triangle

```
const int maxn = ...;
int C[maxn + 1][maxn + 1];
C[0][0] = 1;
for (int n = 1; n <= maxn; ++n) {
   C[n][0] = C[n][n] = 1;
   for (int k = 1; k < n; ++k)
      C[n][k] = C[n - 1][k - 1] + C[n - 1][k];
}
```

3.10 Fft

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
   int n = a.size();
   if (n == 1) return;
   vector<cd> a0(n / 2), a1(n / 2);
   for (int i = 0; 2 * i < n; i++) {
      a0[i] = a[2*i];
      a1[i] = a[2*i+1];
   }
   fft(a0, invert); fft(a1, invert);
   double ang = 2 * PI / n * (invert ? -1 : 1);
   cd w(1), wn(cos(ang), sin(ang));
   for (int i = 0; 2 * i < n; i++) {
      a[i] = a0[i] + w * a1[i];
      a[i + n/2] = a0[i] - w * a1[i];
      if (invert) {
         a[i] /= 2;
         a[i + n/2] /= 2;
      }
      w *= wn;
   }
}
vector<int> multiply(vector<int> const& a, vector<int> const& b) {
   vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
   int n = 1;
   while (n < a.size() + b.size()) n <<= 1;
   fa.resize(n); fb.resize(n);
   fft(fa, false);
   fft(fb, false);
   for (int i = 0; i < n; i++)
      fa[i] *= fb[i];
   fft(fa, true);
   vector<int> result(n);
   for (int i = 0; i < n; i++)
      result[i] = round(fa[i].real());
   return result;
}
void solve() {
   string s, r; cin >> s >> r;
   int n = s.size(), m = r.size();
   char bases[4] = {'A', 'C', 'G', 'T'};
   int minHamming = LLONG_MAX;
   vector<int> hammingDistance(n, 0);
   for (char base : bases) {
      vector<int> sBinary(n, 0), rBinary(m, 0);
      for (int i = 0; i < n; i++)
         sBinary[i] = (s[i] == base);
      for (int i = 0; i < m; i++)
         rBinary[m - i - 1] = !(r[i] == base);
      vector<int> matchCount = multiply(sBinary, rBinary);
      for (int i = m - 1; i < n; i++) {
         int matches = matchCount[i];
         hammingDistance[i] += matches;
      }
   }
   for (int i = m - 1; i < n; i++) {
      minHamming = min(minHamming, hammingDistance[i]);
   }
   cout << minHamming << endl;
}
```

3.11 Binpow

```
int binmult(int a, int b) {
   if (b == 0) {
      return 0;
   }
   int answer = binmult(a, b / 2) % MOD;
   answer = (answer * 2) % MOD;
   if (b & 1) {
      answer = (answer + a) % MOD;
   }
   return answer;
}
int binpow(int x, int n) {
   if (n == 0) {
      return 1;
   } else if (n == 1) {
      return x % MOD;
   }
   int answer = binpow(x, n / 2) % MOD;
   answer = binmult(answer, answer);
   answer %= MOD;
   if (n & 1) {
      answer *= x;
      answer %= MOD;
   }
   return answer;
}
```

4 General

4.1 Template

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#define INF INT_MAX
#define LINF LONG_LONG_MAX
#define int long long
#define all(a) a.begin(), a.end()
#define f first
#define s second
#define vi vector<int>
#define vvi vector<vector<int>>
#define vvvi vector<vector<vector<int>>>
#define vii vector<pair<int, int>>
#define file_read(filepath) freopen(filepath, "r", stdin);
#define file_write(filepath) freopen(filepath, "w", stdout);
#define fastio ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
#define MOD 1000000007
using namespace std;
using pii = pair<int, int>;
using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

template <typename A, typename B>
ostream &operator<<(ostream &os, const pair<A, B> &p) {
   return os << '(' << p.first << ", " << p.second << ')';
}
template <typename T>
void print(vector<T> &array, int size =
    numeric_limits<int>::max()) {
   for (int i = 0; i < min<int>(size, array.size()); i++) {
      cout << array[i] << " ";
   }cout << "\n";
}
template <typename T>
void print(T X) { cout << X << "\n"; }
template <typename T, typename... Ts>
void print(T X, Ts... Y) {
   cout << X << " ";
   print(Y...);
}
```

4.2 Coordinate Compression

```
vector<int> a(n);
vector<int> b = a;
sort(b.begin(), b.end());
```

```
map<int, int> m;
for (int i = 0; i < n; i++) {
   m[b[i]] = i;
}
for (int i = 0; i < n; i++) {
   a[i] = m[a[i]];
}
// Now every value of an array lies in [0, n).
// The most convineint it that if you need the original value for
    a[i],
// you can just write b[a[i]].
```

4.3 Inversion Index

```
int countAndMerge(vector<int> &arr, int l, int m, int r) {
   // Counts in two subarrays
   int n1 = m - l + 1, n2 = r - m;
   vector<int> left(n1), right(n2);
   for (int i = 0; i < n1; i++)
      left[i] = arr[i + l];
   for (int j = 0; j < n2; j++)
      right[j] = arr[m + 1 + j];
   // Initialize inversion count (or result) and merge two halves
   int res = 0;
   int i = 0, j = 0, k = l;
   while (i < n1 && j < n2) {
      if (left[i] <= right[j])
         arr[k++] = left[i++];
      else {
         arr[k++] = right[j++];
         res += (n1 - i);
      }
   }
   while (i < n1)
      arr[k++] = left[i++];
   while (j < n2)
      arr[k++] = right[j++];
   return res;
}
int countInv(vector<int> &arr, int l, int r) {
   int res = 0;
   if (l < r) {
      int m = (r + l) / 2;
      res += countInv(arr, l, m);
      res += countInv(arr, m + 1, r);
      res += countAndMerge(arr, l, m, r);
   }
   return res;
}
int inversionCount(vector<int> &arr) {
   int n = arr.size();
   return countInv(arr, 0, n - 1);
}
```

4.4 Matrix Exponentiation

```
void multiply(vector<vector<int>> &A, vector<vector<int>> B) {
   int n = A.size();
   vector<vector<int>> X(n, vector<int>(n, 0));
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         X[i][j] = 0;
         for (int k = 0; k < n; k++) {
            X[i][j] += (A[i][k] * B[k][j]);
            X[i][j] %= MOD;
         }
      }
   }
   A = X;
}
int n;
cin >> n;
vector<vector<int>> F = {{0, 1}, {1, 1}};
vector<vector<int>> A = {{1, 0}, {0, 1}};
while (n > 0) {
   if (n & 1) {
      multiply(A, F);
   }
   multiply(F, F);
   n >>= 1;
}
multiply(A, {{1, 0}, {1, 0}});
print(A[0][0]);
```

5 Geometry

5.1 Sweep Line

```
const double EPS = 1E-9;
struct pt {
   double x, y;
};
struct seg {
   pt p, q;
   int id;
   double get_y(double x) const {
      if (abs(p.x - q.x) < EPS)
         return p.y;
      return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
   }
};
bool intersect1d(double l1, double r1, double l2, double r2) {
   if (l1 > r1)
      swap(l1, r1);
   if (l2 > r2)
      swap(l2, r2);
   return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt &a, const pt &b, const pt &c) {
   double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x -
       a.x);
   return abs(s) < EPS ? 0 : s > 0 ? +1
                           : -1;
}
bool intersect(const seg &a, const seg &b) {
   return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
        intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
        vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
        vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg &a, const seg &b) {
   double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
   return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
   double x;
   int tp, id;
   event() {}
   event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
   bool operator<(const event &e) const {
      if (abs(x - e.x) > EPS)
         return x < e.x;
      return tp > e.tp;
   }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
   return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
   return ++it;
}
pair<int, int> solve(const vector<seg> &a) {
   int n = (int)a.size();
   vector<event> e;
   for (int i = 0; i < n; ++i) {
      e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
      e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
   }
   sort(e.begin(), e.end());
   s.clear();
   where.resize(a.size());
   for (size_t i = 0; i < e.size(); ++i) {
      int id = e[i].id;
      if (e[i].tp == +1) {
         set<seg>::iterator nxt = s.lower_bound(a[id]), prv =
             prev(nxt);
         if (nxt != s.end() && intersect(*nxt, a[id]))
            return make_pair(nxt->id, id);
         if (prv != s.end() && intersect(*prv, a[id]))
            return make_pair(prv->id, id);
         where[id] = s.insert(nxt, a[id]);
      } else {
         set<seg>::iterator nxt = next(where[id]), prv =
             prev(where[id]);
         if (nxt != s.end() && prv != s.end() && intersect(*nxt,
             *prv))
            return make_pair(prv->id, nxt->id);
         s.erase(where[id]);
      }
   }
```

```
   return make_pair(-1, -1);
}
```

6 Strings

6.1 Aho Corasick

```
using namespace std;
#include <bits/stdc++.h>
const int MAXS = 500;
const int MAXC = 26;
int out[MAXS];
int f[MAXS];
int g[MAXS][MAXC];
int buildMatchingMachine(string arr[], int k) {
   memset(out, 0, sizeof out);
   memset(g, -1, sizeof g);
   int states = 1;
   for (int i = 0; i < k; ++i) {
      const string &word = arr[i];
      int currentState = 0;
      for (int j = 0; j < word.size(); ++j) {
         int ch = word[j] - 'a';
         if (g[currentState][ch] == -1)
            g[currentState][ch] = states++;
         currentState = g[currentState][ch];
      }
      out[currentState] |= (1 << i);
   }
   for (int ch = 0; ch < MAXC; ++ch)
      if (g[0][ch] == -1)
         g[0][ch] = 0;

   memset(f, -1, sizeof f);
   queue<int> q;
   for (int ch = 0; ch < MAXC; ++ch) {
      if (g[0][ch] != 0) {
         f[g[0][ch]] = 0;
         q.push(g[0][ch]);
      }
   }
   while (q.size()) {
      int state = q.front();
      q.pop();
      for (int ch = 0; ch <= MAXC; ++ch) {
         if (g[state][ch] != -1) {
            int failure = f[state];
            while (g[failure][ch] == -1)
               failure = f[failure];
            failure = g[failure][ch];
            f[g[state][ch]] = failure;
            out[g[state][ch]] |= out[failure];
            q.push(g[state][ch]);
         }
      }
   }

   return states;
}

int findNextState(int currentState, char nextInput) {
   int answer = currentState;
   int ch = nextInput - 'a';
   while (g[answer][ch] == -1)
      answer = f[answer];
   return g[answer][ch];
}

void searchWords(string arr[], int k, string text) {
   buildMatchingMachine(arr, k);
   int currentState = 0;
   for (int i = 0; i < text.size(); ++i) {
      currentState = findNextState(currentState, text[i]);
      if (out[currentState] == 0)
         continue;
      for (int j = 0; j < k; ++j) {
         if (out[currentState] & (1 << j)) {
            cout << "Word " << arr[j] << " appears from "
                 << i - arr[j].size() + 1 << " to " << i << endl;
         }
      }
   }
}
int main() {
   string arr[] = {"he", "she", "hers", "his"};
```

```
   string text = "ahishers";
   int k = sizeof(arr) / sizeof(arr[0]);
   searchWords(arr, k, text);
   return 0;
}
```

6.2 Hashing

```
// RNG for hash function
mt19937
    rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
const ll B = uniform_int_distribution<ll>(0, M - 1)(rng);
class HashedString {
public:
   // change M and B if you want
   static const long long M = 1e9 + 9; // 2^61 - 1
   static const long long B = 9973;
   // pow[i] contains B^i % M
   static vector<long long> pow;
   // p_hash[i] is the hash of the first i characters of the given
       string
   vector<long long> p_hash;
   HashedString(const string &s) : p_hash(s.size() + 1) {
      while (pow.size() <= s.size()) {
         pow.push_back((pow.back() * B) % M);
      }
      p_hash[0] = 0;
      for (int i = 0; i < s.size(); i++) {
         p_hash[i + 1] = ((p_hash[i] * B) % M + s[i]) % M;
      }
   }
   long long get_hash(int start, int end) {
      long long raw_val = (p_hash[end + 1] - (p_hash[start] *
          pow[end - start + 1]));
      return (raw_val % M + M) % M;
   }
};
vector<long long> HashedString::pow = {1};
```

6.3 Trie

```
class Trie {
public:
   Trie *child[26];
   bool worldEnd;
   Trie() {
      for (int i = 0; i < 26; i++) {
         child[i] = nullptr;
      }
      worldEnd = false;
   }
};
void insert(Trie *root, string s) {
   int n = s.size();
   Trie *current = root;
   for (int i = 0; i < n; i++) {
      if (current->child[s[i] - 'a'] == nullptr) {
         current->child[s[i] - 'a'] = new Trie();
      }
      current = current->child[s[i] - 'a'];
   }
   current->worldEnd = true;
}
bool search(Trie *root, string s) {
   int n = s.size();
   Trie *current = root;
   for (int i = 0; i < n; i++) {
      if (current->child[s[i] - 'a'] == nullptr) {
         return false;
      }
      current = current->child[s[i] - 'a'];
   }
   return current->worldEnd;
}
void solve() {
   Trie *root = new Trie();
   vector<string> arr =
      {"and", "ant", "do", "geek", "dad", "ball"};
   for (const string &s : arr) {
      insert(root, s);
   }
   // One by one search strings
   vector<string> searchKeys = {"do", "gee", "bat"};
```

```
   for (string &s : searchKeys) {
      cout << "Key : " << s << "\n";
      if (search(root, s))
         cout << "Present\n";
      else
         cout << "Not Present\n";
   }
}
```

6.4 Rabin Karp

```
vector<int> rabin_karp(string const &s, string const &t) {
   const int p = 31;
   const int m = 1e9 + 9;
   int S = s.size(), T = t.size();

   vector<long long> p_pow(max(S, T));
   p_pow[0] = 1;
   for (int i = 1; i < (int)p_pow.size(); i++)
      p_pow[i] = (p_pow[i - 1] * p) % m;

   vector<long long> h(T + 1, 0);
   for (int i = 0; i < T; i++)
      h[i + 1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
   long long h_s = 0;
   for (int i = 0; i < S; i++)
      h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

   vector<int> occurrences;
   for (int i = 0; i + S - 1 < T; i++) {
      long long cur_h = (h[i + S] + m - h[i]) % m;
      if (cur_h == h_s * p_pow[i] % m)
         occurrences.push_back(i);
   }
   return occurrences;
}
```

6.5 Minimum String Rotation

```
int minimumExpression(string s) {
  s = s + s;
  int len = s.size(), i = 0, j = 1, k = 0;
  while (i + k < len && j + k < len) {
    if (s[i + k] == s[j + k])
      k++;
    else if (s[i + k] > s[j + k]) {
      i = i + k + 1;
      if (i <= j) i = j + 1;
      k = 0;
    } else if (s[i + k] < s[j + k]) {
      j = j + k + 1;
      if (j <= i) j = i + 1;
      k = 0;
    }
  }
  return min(i, j);
}
```

6.6 Z Function

```
vector<int> z_function(string s) {
   int n = s.size();
   vector<int> z(n);
   int l = 0, r = 0;
   for (int i = 1; i < n; i++) {
      if (i < r) {
         z[i] = min(r - i, z[i - l]);
      }
      while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
         z[i]++;
      }
      if (i + z[i] > r) {
         l = i;
         r = i + z[i];
      }
   }
   return z;
}
```

7 Data Structures

7.1 Lca

```
vvi adj; vi depth;
void dfs(int v, int p, int d) {
   depth[v] = d;
   for (auto u : adj[v]) {
      if (u != p) { dfs(u, v, d + 1); }
   }
}
depth.assign(n, 0); adj.assign(n, vi());
int maxbit = 21;
vector<int> parent(n, -1);
vvi ancestor(maxbit + 1, vi(n, -1));
for (int i = 1; i < n; i++) {
   cin >> u; u--;
   parent[i] = u; ancestor[0][i] = u;
   adj[u].push_back(i); adj[i].push_back(u);
}
dfs(0, 0, 0);
for (int i = 1; i <= maxbit; i++) {
   for (int j = 0; j < n; j++) {
      if (ancestor[i - 1][j] == -1) continue;
      ancestor[i][j] = ancestor[i - 1][ancestor[i - 1][j]];
   }
}
// LCA of a and b
if (depth[a] > depth[b])
   swap(a, b);
int diff = depth[b] - depth[a];
for (int j = maxbit; j >= 0; j--) {
   if ((1 << j) <= diff) {
      b = ancestor[j][b];
      diff -= (1 << j);
   }
}
if (a == b) { print(a + 1); }
else{ for (int j = maxbit; j >= 0; j--) {
   if (ancestor[j][a] != -1 && (ancestor[j][a] != ancestor[j][b]))
       {
      a = ancestor[j][a]; b = ancestor[j][b];
   }
}
print(ancestor[0][a] + 1); }
```

7.2 Dsu

```
class DSU {
public:
   int n; vector<int> parent, size;
   DSU(int n) {
      this->n = n; parent.assign(n, -1); size.assign(n, 0);
   }
   void make_set(int v) {
      parent[v] = v; size[v] = 1;
   }
   int find_set(int v) {
      if (v == parent[v]) { return v; }
      return parent[v] = find_set(parent[v]);
   }
   void union_set(int a, int b) {
      a = find_set(a); b = find_set(b);
      if (a != b) {
         if (size[a] < size[b]) swap(a, b);
         parent[b] = a; size[a] += size[b];
      }
   }
   int total_sets() {
      int total = 0;
      for (int i = 1; i < parent.size(); i++)
      { if (i == parent[i]) total++; }
      return total;
   }
};
```

7.3 Segment Tree Walk

```
class SegmentTree {
public:
   int n;
   vii tree;
```

```
SegmentTree(vi &array) {
   n = array.size();
   tree.assign(4 * n, pii());
   build(array, 1, 0, n - 1);
}
pii f(pii a, pii b) {
   if (a.f > b.f) {
      return a;
   } else if (a.f < b.f) {
      return b;
   } else {
      return pii(a.f, a.s + b.s);
   }
}
void build(vi &array, int v, int tl, int tr) {
   if (tl == tr) {
      tree[v] = {array[tl], 1};
   } else {
      int tm = midpoint(tl, tr);
      build(array, 2 * v, tl, tm);
      build(array, 2 * v + 1, tm + 1, tr);
      tree[v] = f(tree[2 * v], tree[2 * v + 1]);
   }
}
void update(int v, int tl, int tr, int pos, int val) {
   if (tl == tr) {
      tree[v].f -= val;
   } else {
      int tm = midpoint(tl, tr);
      if (pos <= tm) {
         update(2 * v, tl, tm, pos, val);
      } else {
         update(2 * v + 1, tm + 1, tr, pos, val);
      }
      tree[v] = f(tree[2 * v], tree[2 * v + 1]);
   }
}
pii conquer(int v, int tl, int tr, int l, int r) {
   if (l > r) {
      return {-INF, -INF};
   } else if (l == tl && r == tr) {
      return tree[v];
   } else {
      int tm = midpoint(tl, tr);
      pii answer = f(conquer(2 * v, tl, tm, l, min(tm, r)),
          conquer(2 * v + 1, tm + 1, tr, max(l, tm + 1), r));
      return answer;
   }
}
int walk(int v, int tl, int tr, int needed) {
   if (needed > tree[v].first) {
      return -1;
   }
   if (tl == tr) {
      return tl;
   }
   int tm = midpoint(tl, tr);
   if (needed <= tree[2 * v].f) {
      return walk(2 * v, tl, tm, needed);
   } else {
      return walk(2 * v + 1, tm + 1, tr, needed);
   }
}
};
```

7.4 Segment Tree Lazy

```
class SegmentTree {
public:
   int n;
   vi tree, lazy;
   SegmentTree(vi &array) {
      n = array.size();
      tree.assign(4 * n, 0);
      lazy.assign(4 * n, 0);
      build(array, 1, 0, n - 1);
   }
   int f(int a, int b) {
      return a + b;
   }
   void build(vi &array, int v, int tl, int tr) {
      if (tl == tr) {
         tree[v] = array[tl];
      } else {
         int tm = (tl + tr) / 2;
         build(array, v * 2, tl, tm);
```

```
        build(array, v * 2 + 1, tm + 1, tr);
        tree[v] = f(tree[v * 2], tree[v * 2 + 1]);
    }
}
void updateRange(int v, int tl, int tr, int l, int r, int
     value) {
    if (lazy[v] != 0) {
        tree[v] += (tr - tl + 1) * lazy[v];
        if (tl != tr) {
            lazy[v * 2] += lazy[v];
            lazy[v * 2 + 1] += lazy[v];
        }
        lazy[v] = 0;
    }
    if (l > r) {
        return;
    }
    if (l == tl && r == tr) {
        tree[v] += (tr - tl + 1) * value;
        if (tl != tr) {
            lazy[v * 2] += value;
            lazy[v * 2 + 1] += value;
        }
        return;
    }
    // int tm = midpoint(tl, tr);
    int tm = (tl + tr) / 2;
    updateRange(v * 2, tl, tm, l, min(tm, r), value);
    updateRange(v * 2 + 1, tm + 1, tr, max(l, tm + 1), r, value);
    tree[v] = f(tree[v * 2], tree[v * 2 + 1]);
}
int query(int v, int tl, int tr, int l, int r) {
    if (lazy[v] != 0) {
        tree[v] += (tr - tl + 1) * lazy[v];
        if (tl != tr) {
            lazy[v * 2] += lazy[v];
            lazy[v * 2 + 1] += lazy[v];
        }
        lazy[v] = 0;
    }
    if (l > r) {
        return 0;
    }
    if (l == tl && r == tr) {
        return tree[v];
    }
    int tm = (tl + tr) / 2;
    return f(query(v * 2, tl, tm, l, min(tm, r)), query(v * 2 +
        1, tm + 1, tr, max(l, tm + 1), r));
  }
};
```

Number of divisors:

$$d(n) = (e_1 + 1) \cdot (e_2 + 1) \cdots (e_k + 1)$$

Sum of divisors:

$$\sigma(n) = \frac{p_1^{e_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{e_2+1} - 1}{p_2 - 1} \cdots \frac{p_k^{e_k+1} - 1}{p_k - 1}$$

Substring hashing: When polynomial hashing is used, we can calculate the hash value of any substring of a string s in $O(1)$ time after an $O(n)$ time preprocessing. The idea is to construct an array h such that $h[k]$ contains the hash value of the prefix $s[0 \ldots k]$. The array values can be recursively calculated as follows:

$$h[0] = s[0]$$

$$h[k] = (h[k-1]A + s[k]) \mod B$$

In addition, we construct an array p where $p[k] = A^k \mod B$:

$$p[0] = 1$$

$$p[k] = (p[k-1]A) \mod B.$$

Constructing the above arrays takes $O(n)$ time. After this, the hash value of any substring $s[a \ldots b]$ can be calculated in $O(1)$ time using the formula:

$$(h[b] - h[a-1]p[b-a+1]) \mod B$$

assuming that $a > 0$. If $a = 0$, the hash value is simply $h[b]$.