Ucs

```
#include <bits/stdc++.h>
using namespace std;

map<char, vector<pair<char,int>>> adj;
map<char, bool> vis;
char en;

void ucs(char start) {
    // priority_queue: pair<cost, path>
    priority_queue<pair<int, vector<char>>,
                   vector<pair<int, vector<char>>>,
                   greater<pair<int, vector<char>>>> pq;

    pq.push({0, vector<char>{start}}); // initial cost 0, path
with start

    while (!pq.empty()) {
        int cost = pq.top().first;
        vector<char> path = pq.top().second;
        pq.pop();

        char node = path.back();

        if (vis[node]) continue;
        vis[node] = true;
```

```
        cout << node << " " << cost << endl;

        if (node == en) {
            cout << "Path: ";
            for (char c : path) cout << c;
            cout << "\nTotal cost: " << cost << endl;
            return;
        }

        for (auto v : adj[node]) {
            char next = v.first;
            int w = v.second;
            if (!vis[next]) {
                vector<char> new_path = path;
                new_path.push_back(next);
                pq.push({cost + w, new_path});
            }
        }
    }

    cout << "Not found\n";
}

int main() {
    int n, w;
    cin >> n;
```

```
    char start;
    for (int i = 0; i < n; i++) {
        char u, v;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w}); // undirected
    }

    cin >> start >> en;

    cout << "UCS traversal:\n";
    ucs(start);

    return 0;
}
```

Ids:

```
#include <bits/stdc++.h>
using namespace std;

map<char, vector<char>> adj;
map<char, bool> vis;
char en;

// Depth-Limited Search
bool dls(char u, int depth, int limit) {
    if (depth > limit) return false;
```

```
    vis[u] = true;
    cout << u << "->";

    if (u == en) {
        cout << "\nReached " << en << " at depth " << depth
<< endl;
        return true;
    }

    for (char v : adj[u]) {
        if (!vis[v]) {
            if (dls(v, depth + 1, limit)) return true;
        }
    }

    return false;
}

// Iterative Deepening Search
void ids(char start, int maxd) {
    for (int depth = 0; depth <= maxd; depth++) {
        // reset visited
        for (auto &p : vis) p.second = false;

        cout << "\nTrying depth = " << depth << endl;
        if (dls(start, 0, depth)) return;
    }
```

```
    cout << "Did not find " << en << " within max depth " <<
maxd << endl;
}

int main() {
    int n;
    cin >> n;               // number of edges

    char st;
    int maxd;
    cin >> st >> en >> maxd;

    // read edges
    for (int i = 0; i < n; i++) {
        char u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
        vis[u] = vis[v] = false;
    }

    ids(st, maxd);

    return 0;
}
```

DLA:

```
#include<bits/stdc++.h>
using namespace std;
map<char,vector<char>>adj;
map<char,bool>vis;
char target;
bool found=false;
void dls(char u, int depth, int limit){
    if (depth > limit || found)return;
    vis[u]=true;
    cout << u <<"->"<< " ";
    if(u==target){
        found = true;
        cout<<"\nReached " <<target <<" at depth
"<<depth<<endl;
        return;
    }
    for(char v:adj[u]){
        if (!vis[v]){
            dls(v, depth + 1,limit);
            if(found)return;
        }
    }
}

int main(){
    int n;
```

```
    cin>>n;
    for(int i=0;i<n;i++){
        char u,v;
        cin>>u>>v;
        adj[u].push_back(v);
        adj[v].push_back(u);
        vis[u]=false;
        vis[v]=false;
    }
    char start,en;
    int limit;
    cin >>start>>en>>limit;
    target=en;
    cout<<"DLS "<<limit<<": ";
    dls(start, 0,limit);
    if(!found)
        cout<<"\nDid not reach " <<target <<" within depth
"<<limit<<endl;

    return 0;
}
```

Dfs

```
#include <bits/stdc++.h>
using namespace std;

map<char, vector<char>> adj;
map<char, bool> vis;
```

```
char en;

bool dfs(char start) {
    stack<char> st;       // create a stack for DFS
    st.push(start);       // push the starting node onto the
stack
    vis[start] = true;    // mark start as visited

    while (!st.empty()) {
        char u = st.top();
        st.pop();
        cout << u << " "; // print node as visited

        if (u == en) return true; // reached target

        for (auto it = adj[u].rbegin(); it != adj[u].rend(); ++it) {
    char v = *it;
    if (!vis[v]) {
        vis[v] = true;
        st.push(v);
    }
}

    }
    return false; // target not reached
}
```

```
int main() {
    int m;
    cin >> m;

    for (int i = 0; i < m; i++) {
        char u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
        vis[u] = vis[v] = false;
    }

    char start;
    cin >> start >> en;

    if (dfs(start))
        cout << "\nReach\n";
    else
        cout << "\nNot Reach\n";
}
```

Bfs

```
#include <bits/stdc++.h>
using namespace std;

map<char, vector<char>> adj;
map<char, bool> vis;
char en;
```

```
map<int, vector<char>> ln;
map<char, int> level;
map<char,char>parent;
void bfs(char st) {
    queue<char> q;
    vis[st] = true;
    level[st] = 0;
    parent[st]='\0';
    q.push(st);

    while (!q.empty()) {
        char u = q.front();
        q.pop();
        cout << u << "->";
        ln[level[u]].push_back(u);

        for (char v : adj[u]) {
            if (!vis[v]) {
                vis[v] = true;
                parent[v]=u;
                q.push(v);
                level[v] = level[u] + 1;

            }
        }
    }
}
```

```
int main() {
    int n;
    char st;
    cin >> n;

    for (int i = 0; i < n; i++) {
        char u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
        vis[u] = vis[v] = false;
    }

    cin >> st >> en;
    bfs(st);

    if (level.count(en))
        cout << "\nLevel of " << en << ": " << level[en] << "\n";
    else
        cout << "\nTarget not reachable\n";

    cout << "\n--- Nodes by level ---\n";
    for (auto n : ln) {
        cout << n.first << " -> ";
        for (auto nd : n.second) cout << nd << " ";
        cout << "\n";
```

```
    }

    vector<char> path;
    for (char v =en;v!= '\0'; v = parent[v])
        path.push_back(v);
    reverse(path.begin(), path.end());

    cout << "Path from start to " <<en << ": ";
    for (int i = 0; i < path.size(); i++) {
        cout << path[i];

    }
    cout << "\n";
}
```

A*

```
#include <bits/stdc++.h>
using namespace std;

map<char, vector<pair<char,int>>> graph;
map<char,int> heuristic;
char goal;

void aStar(char start) {
    priority_queue<pair<int, vector<char>>, vector<pair<int,
vector<char>>>, greater<>> pq;
    map<char,bool> visited;
```

```
    map<char,int> g;

    g[start] = 0;
    pq.push({heuristic[start], {start}});

    while(!pq.empty()) {
       pair<int, vector<char>> top = pq.top();
pq.pop();
int f_cost = top.first;
vector<char> path = top.second;

char current = path.back();
;

        if(visited[current]) continue;
        visited[current] = true;

        cout << "Visiting: " << current << " with f = " << f_cost
<< endl;

        if(current == goal) {
            cout << "\nGoal Reached! Path: ";
            for(char node : path) cout << node << " ";
            cout << "\nTotal cost: " << g[current] << endl;
            return;
        }
```

```
    for(int i = 0; i < graph[current].size(); i++) {
    char neighbor = graph[current][i].first;
    int cost = graph[current][i].second;

            if(!visited[neighbor]) {
                int new_g = g[current] + cost;
                int new_f = new_g + heuristic[neighbor];
                if(!g.count(neighbor) || new_g < g[neighbor]) {
                    g[neighbor] = new_g;
                    auto new_path = path;
                    new_path.push_back(neighbor);
                    pq.push({new_f, new_path});
                }
            }
        }
    }

    cout << "Goal not reachable.\n";
}

int main() {
    int edges, h_values;
    char start;

    cout << "Enter number of edges: ";
    cin >> edges;
    cout << "Enter edges (node1 node2 cost):\n";
```

```
    for(int i = 0; i < edges; i++) {
        char u, v; int cost;
        cin >> u >> v >> cost;
        graph[u].push_back({v, cost});
        graph[v].push_back({u, cost});
    }

    cout << "Enter number of heuristic values: ";
    cin >> h_values;
    cout << "Enter heuristics (node value):\n";
    for(int i = 0; i < h_values; i++) {
        char node; int h;
        cin >> node >> h;
        heuristic[node] = h;
    }

    cout << "Enter start node: ";
    cin >> start;
    cout << "Enter goal node: ";
    cin >> goal;

    cout << "\nA* Search Path:\n";
    aStar(start);

    return 0;
}
```

8puzzle

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
    vector<vector<int>> puzzle;
    int g, h;
    string path;
    bool operator>(const Node &other) const {
        return g + h > other.g + other.h;
    }
};

vector<vector<int>> goal = {{1,2,3},{4,5,6},{7,8,0}};

bool isGoal(const vector<vector<int>>& p) {
    return p == goal;
}

int heuristic(const vector<vector<int>>& p) {
    int dist = 0;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (p[i][j] != 0) {
                int val = p[i][j] - 1;
                dist += abs(i - val/3) + abs(j - val%3);
            }
    return dist;
```

```
}

pair<int,int> findZero(const vector<vector<int>>& p) {
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (p[i][j] == 0) return {i,j};
    return {0,0};
}

vector<Node> getNext(const Node &cur) {
    vector<Node> res;
    int dx[4] = {-1,1,0,0};
    int dy[4] = {0,0,-1,1};
    char moves[4] = {'U','D','L','R'};
    pair<int,int> zeroPos = findZero(cur.puzzle);
    int x = zeroPos.first, y = zeroPos.second;
    for (int k = 0; k < 4; k++) {
        int nx = x + dx[k], ny = y + dy[k];
        if (nx>=0 && nx<3 && ny>=0 && ny<3) {
            Node next = cur;
            swap(next.puzzle[x][y], next.puzzle[nx][ny]);
            next.g = cur.g + 1;
            next.h = heuristic(next.puzzle);
            next.path = cur.path + moves[k];
            res.push_back(next);
        }
    }
```

```
    return res;
}

void solve(vector<vector<int>> start) {
    priority_queue<Node, vector<Node>, greater<Node>>
pq;
    set<vector<vector<int>>> visited;
    Node sn = {start, 0, heuristic(start), ""};
    pq.push(sn);
    while(!pq.empty()) {
        Node cur = pq.top(); pq.pop();
        if (isGoal(cur.puzzle)) {
            cout << "Goal reached!\n";
            cout << "Moves: " << cur.g << "\n";
            cout << "Path: " << cur.path << endl;
            return;
        }
        if (visited.count(cur.puzzle)) continue;
        visited.insert(cur.puzzle);
        for (auto next : getNext(cur)) {
            if (!visited.count(next.puzzle)) pq.push(next);
        }
    }
    cout << "No solution found.\n";
}

int main() {
```

```
    vector<vector<int>> start(3, vector<int>(3));
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cin >> start[i][j];
    solve(start);
}
```

Hill climb

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
    vector<vector<int>> puzzle;
    int h;
    string path;
};

vector<vector<int>> goal = {{1,2,3},{4,5,6},{7,8,0}};

bool isGoal(const vector<vector<int>>& p) {
    return p == goal;
}

int heuristic(const vector<vector<int>>& p) {
    int dist = 0;
    for (int i = 0; i < 3; i++)
```

```
        for (int j = 0; j < 3; j++)
            if (p[i][j] != 0) {
                int val = p[i][j] - 1;
                dist += abs(i - val / 3) + abs(j - val % 3);
            }
    return dist;
}

pair<int,int> findZero(const vector<vector<int>>& p) {
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (p[i][j] == 0) return {i, j};
    return {0, 0};
}

vector<Node> getNext(const Node &cur) {
    vector<Node> res;
    int dx[4] = {-1, 1, 0, 0};
    int dy[4] = {0, 0, -1, 1};
    char mv[4] = {'U', 'D', 'L', 'R'};
    auto [x, y] = findZero(cur.puzzle);
    for (int k = 0; k < 4; k++) {
        int nx = x + dx[k], ny = y + dy[k];
        if (nx >= 0 && nx < 3 && ny >= 0 && ny < 3) {
            Node next = cur;
            swap(next.puzzle[x][y], next.puzzle[nx][ny]);
            next.h = heuristic(next.puzzle);
```

```
            next.path = cur.path + mv[k];
            res.push_back(next);
        }
    }
    return res;
}

void printPuzzle(const vector<vector<int>> &p) {
    for (auto &r : p) {
        for (int v : r) cout << v << " ";
        cout << "\n";
    }
}

void hillClimb(vector<vector<int>> start) {
    Node cur = {start, heuristic(start), ""};
    cout << "Initial heuristic: " << cur.h << endl;

    while (true) {
        if (isGoal(cur.puzzle)) {
            cout << "\nGoal reached!\n";
            cout << "Moves: " << cur.path.size() << "\n";
            cout << "Path: " << cur.path << "\n";
            return;
        }

        vector<Node> nextStates = getNext(cur);
```

```
        Node best = cur;
        for (auto &n : nextStates)
            if (n.h < best.h)
                best = n;

        if (best.h >= cur.h) {
            cout << "\nReached local minimum or plateau.\n";
            cout << "Final heuristic: " << cur.h << "\n";
            cout << "Path: " << cur.path << "\n";
            return;
        }

        cout << "Current h = " << cur.h
             << " -> Next h = " << best.h
             << " | Move: " << best.path.back() << endl;

        cur = best;
    }
}

int main() {
    vector<vector<int>> start(3, vector<int>(3));
    cout << "Enter puzzle (use 0 for blank):\n";
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cin >> start[i][j];
    hillClimb(start);
```

```
}
```

Tictactoe

```
#include <bits/stdc++.h>
using namespace std;

char board[3][3];
const char HUMAN = 'X', AI = 'O';

// Print board
void printBoard() {
    for (int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            cout << (board[i][j]==' ' ? '_' : board[i][j]);
            if(j<2) cout<<"|";
        }
        cout<<"\n";
    }
    cout<<"\n";
}

// Check for win
int evaluate() {
    // Rows & columns
    for(int i=0;i<3;i++){
        if(board[i][0]==board[i][1] && board[i][1]==board[i][2]){
            if(board[i][0]==AI) return +10;
```

```
            if(board[i][0]==HUMAN) return -10;
        }
        if(board[0][i]==board[1][i] && board[1][i]==board[2][i]){
            if(board[0][i]==AI) return +10;
            if(board[0][i]==HUMAN) return -10;
        }
    }
    // Diagonals
    if(board[0][0]==board[1][1] && board[1][1]==board[2][2]){
        if(board[0][0]==AI) return +10;
        if(board[0][0]==HUMAN) return -10;
    }
    if(board[0][2]==board[1][1] && board[1][1]==board[2][0]){
        if(board[0][2]==AI) return +10;
        if(board[0][2]==HUMAN) return -10;
    }
    return 0;
}

// Check moves left
bool movesLeft() {
    for(auto &r: board)
        for(char c:r)
            if(c==' ') return true;
    return false;
}
```

```
// Minimax with alpha-beta
int minimax(bool isMax, int alpha, int beta) {
    int score = evaluate();
    if(score==10 || score==-10) return score;
    if(!movesLeft()) return 0;

    if(isMax){
        int best=-1000;
        for(int i=0;i<3;i++)
            for(int j=0;j<3;j++)
                if(board[i][j]==' '){
                    board[i][j]=AI;
                    best = max(best, minimax(false, alpha, beta));
                    board[i][j]=' ';
                    alpha = max(alpha,best);
                    if(beta<=alpha) return best;
                }
        return best;
    } else {
        int best=1000;
        for(int i=0;i<3;i++)
            for(int j=0;j<3;j++)
                if(board[i][j]==' '){
                    board[i][j]=HUMAN;
                    best = min(best, minimax(true, alpha, beta));
                    board[i][j]=' ';
                    beta = min(beta,best);
```

```
                if(beta<=alpha) return best;
            }
        return best;
    }
}

// Best AI move
pair<int,int> bestMove() {
    int bestVal=-1000;
    pair<int,int> move = {-1,-1};
    for(int i=0;i<3;i++)
        for(int j=0;j<3;j++)
            if(board[i][j]==' '){
                board[i][j]=AI;
                int val = minimax(false, -1000, 1000);
                board[i][j]=' ';
                if(val>bestVal){
                    bestVal=val;
                    move={i,j};
                }
            }
    return move;
}

// Main game
int main() {
    for(auto &r:board) fill(begin(r), end(r),' ');
```

```
    printBoard();
    while(true){
        int pos;
        cout<<"Enter position (1-9): ";
        cin>>pos;
        int r=(pos-1)/3, c=(pos-1)%3;
        if(pos<1||pos>9 || board[r][c]!=' '){ cout<<"Invalid!\n";
continue;}
        board[r][c]=HUMAN;
        printBoard();
        if(evaluate()==-10){ cout<<"You win!\n"; break;}
        if(!movesLeft()){ cout<<"Draw!\n"; break;}

        cout<<"AI is thinking...\n";
        pair<int,int>het=bestMove();
        int i=het.first;
        int j=het.second;
        board[i][j]=AI;
        printBoard();
        if(evaluate()==10){ cout<<"AI wins!\n"; break;}
        if(!movesLeft()){ cout<<"Draw!\n"; break;}
    }
}
```

Without prune

```
#include <bits/stdc++.h>
```

```
using namespace std;

char BOARD[3][3];
const char AI = 'X';
const char HU = 'O';

void print() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << (BOARD[i][j] == ' ' ? '_' : BOARD[i][j]);
            if (j < 2) cout << '|';
        }
        cout << endl;
    }
    cout << endl;
}

bool movesLeft() {
    for (auto &r : BOARD)
        for (auto c : r)
            if (c == ' ') return true;
    return false;
}

int evaluate() {
    for (int i = 0; i < 3; i++) {
        // Row check
```

```
        if (BOARD[i][0] == BOARD[i][1] && BOARD[i][1] ==
BOARD[i][2] && BOARD[i][0] != ' ') {
            if (BOARD[i][0] == AI) return +10;
            else return -10;
        }
        // Column check
        if (BOARD[0][i] == BOARD[1][i] && BOARD[1][i] ==
BOARD[2][i] && BOARD[0][i] != ' ') {
            if (BOARD[0][i] == AI) return +10;
            else return -10;
        }
    }
    // Diagonal check
    if (BOARD[0][0] == BOARD[1][1] && BOARD[1][1] ==
BOARD[2][2] && BOARD[0][0] != ' ') {
        if (BOARD[0][0] == AI) return +10;
        else return -10;
    }
    if (BOARD[0][2] == BOARD[1][1] && BOARD[1][1] ==
BOARD[2][0] && BOARD[0][2] != ' ') {
        if (BOARD[0][2] == AI) return +10;
        else return -10;
    }
    return 0;
}

int minimax(bool isMax) {
```

```
int score = evaluate();

if (score == 10 || score == -10) return score;
if (!movesLeft()) return 0;

if (isMax) {  // AI's move
    int best = -1000;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (BOARD[i][j] == ' ') {
                BOARD[i][j] = AI;
                best = max(best, minimax(false));
                BOARD[i][j] = ' ';
            }
        }
    }
    return best;
} else {  // Human's move
    int best = 1000;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (BOARD[i][j] == ' ') {
                BOARD[i][j] = HU;
                best = min(best, minimax(true));
                BOARD[i][j] = ' ';
            }
        }
```

```
        }
        return best;
    }
}

pair<int, int> bestMove() {
    int bestVal = -1000;
    pair<int, int> move = {-1, -1};
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (BOARD[i][j] == ' ') {
                BOARD[i][j] = AI;
                int val = minimax(false);
                BOARD[i][j] = ' ';
                if (val > bestVal) {
                    bestVal = val;
                    move = {i, j};
                }
            }
        }
    }
    return move;
}

int main() {
    for (auto &r : BOARD) fill(begin(r), end(r), ' ');
    print();
```

```
    while (true) {
        int mv;
        cin >> mv;
        int r = (mv - 1) / 3, c = (mv - 1) % 3;
        if (mv < 1 || mv > 9 || BOARD[r][c] != ' ') {
            cout << "Invalid move!\n";
            continue;
        }
        BOARD[r][c] = HU;
        print();

        if (evaluate() == -10) { cout << "Human wins!\n";
break; }
        if (!movesLeft()) { cout << "Draw!\n"; break; }

        cout << "AI is thinking...\n";
        pair<int, int> gen = bestMove();
        BOARD[gen.first][gen.second] = AI;
        print();

        if (evaluate() == 10) { cout << "AI wins!\n"; break; }
        if (!movesLeft()) { cout << "Draw!\n"; break; }
    }
}
```

Huristic

```
#include <bits/stdc++.h>
using namespace std;

char board[3][3];
const char HUMAN='X', AI='O';

// Print board
void printBoard() {
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            cout << (board[i][j]==' ' ? '_' : board[i][j]);
            if(j<2) cout<<"|";
        }
        cout << endl;
    }
    cout << endl;
}

// Check winner
char checkWinner() {
    for(int i=0;i<3;i++){
        if(board[i][0]==board[i][1] && board[i][1]==board[i][2])
return board[i][0];
        if(board[0][i]==board[1][i] && board[1][i]==board[2][i])
return board[0][i];
    }
```

```
    if(board[0][0]==board[1][1] && board[1][1]==board[2][2])
return board[0][0];
    if(board[0][2]==board[1][1] && board[1][1]==board[2][0])
return board[0][2];
    return ' ';
}

// Check moves left
bool movesLeft() {
    for(auto &r:board) for(auto &c:r) if(c==' ') return true;
    return false;
}

// Heuristic AI: win -> block -> center -> corner -> any
pair<int,int> findBestMove() {
    //1 Win if possible
    for(int i=0;i<3;i++)
        for(int j=0;j<3;j++)
            if(board[i][j]==' '){
                board[i][j]=AI;
                if(checkWinner()==AI) return {i,j};
                board[i][j]=' ';
            }
    //2 Block human
    for(int i=0;i<3;i++)
        for(int j=0;j<3;j++)
            if(board[i][j]==' '){
```

```
            board[i][j]=HUMAN;
            if(checkWinner()==HUMAN){ board[i][j]=' ';
return {i,j}; }
            board[i][j]=' ';
        }
    //3 Take center
    if(board[1][1]==' ') return {1,1};
    //4 Take corner
    vector<pair<int,int>> corners={{0,0},{0,2},{2,0},{2,2}};
    for(auto c:corners) if(board[c.first][c.second]==' ') return
c;
    //5 Take any move
    for(int i=0;i<3;i++) for(int j=0;j<3;j++) if(board[i][j]==' ')
return {i,j};
    return {-1,-1};
}

// Main game
int main() {
    for(auto &r:board) fill(begin(r),end(r),' ');
    printBoard();
    while(true){
        int pos;
        cout<<"Enter position (1-9): ";
        cin>>pos;
        if(pos<1 || pos>9) { cout<<"Invalid!\n"; continue; }
        int r=(pos-1)/3, c=(pos-1)%3;
```

```
        if(board[r][c]!=' '){ cout<<"Invalid!\n"; continue; }
        board[r][c]=HUMAN;
        printBoard();
        if(checkWinner()==HUMAN){ cout<<"You win!\n";
break; }
        if(!movesLeft()){ cout<<"Draw!\n"; break; }

        cout<<"AI is thinking...\n";
        pair<int,int> ai=findBestMove();
        board[ai.first][ai.second]=AI;
        printBoard();
        if(checkWinner()==AI){ cout<<"AI wins!\n"; break; }
        if(!movesLeft()){ cout<<"Draw!\n"; break; }
    }
}
```

N QUEEN

```
#include <bits/stdc++.h>
using namespace std;

vector<vector<int>> bd;
int n;
int sol = 0;

bool check(int r,int c){
    for(int i=0;i<r;i++)
```

```
        if(bd[i][c]==1) return false;
    for(int i=r-1,j=c-1;i>=0&&j>=0;i--,j--)
        if(bd[i][j]==1) return false;
    for(int i=r-1,j=c+1;i>=0&&j<n;i--,j++)
        if(bd[i][j]==1) return false;
    return true;
}

void solve(int row){
    if(row==n){
        sol++;
        cout << "Solution " << sol << ":\n";
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++)
                cout << (bd[i][j] ? "Q" : ".") << " ";
            cout << endl;
        }
        cout << "Queen positions:\n";
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                if(bd[i][j])
                    cout << i+1 << "->" << j+1 << endl;
        cout << endl;
        return;
    }

    for(int col=0;col<n;col++){
```

```
        if(check(row,col)){
            bd[row][col]=1;
            solve(row+1);
            bd[row][col]=0;
        }
    }
}

int main(){
    cin >> n;
    bd.assign(n, vector<int>(n,0));
    solve(0);
    cout << "Total solutions: " << sol << endl;
}
```

Genetic algo

```
#include <bits/stdc++.h>
using namespace std;

struct Individual {
    vector<int> genes; // row->column mapping
    int fitness;
};
```

```
// Parameters
int n = 8; // board size
int population_size = 100;
int max_generations = 1000;
double mutation_rate = 0.1;
vector<Individual> population;

// Fitness function: count non-attacking pairs
int fitness(const vector<int>& genes) {
    int attacks = 0;
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(genes[i]==genes[j] ||
abs(genes[i]-genes[j])==abs(i-j)) attacks++;
        }
    }
    return (n*(n-1)/2 - attacks);
}

// Create random individual
Individual randomIndividual() {
    Individual ind;
    ind.genes.resize(n);
    for(int i=0;i<n;i++) ind.genes[i]=rand()%n;
    ind.fitness = fitness(ind.genes);
    return ind;
}
```

```
// Crossover: single point
Individual crossover(const Individual &p1, const Individual
&p2) {
    Individual child;
    child.genes.resize(n);
    int point = rand()%n;
    for(int i=0;i<point;i++) child.genes[i]=p1.genes[i];
    for(int i=point;i<n;i++) child.genes[i]=p2.genes[i];
    child.fitness = fitness(child.genes);
    return child;
}

// Mutation: random change
void mutate(Individual &ind) {
    if((double)rand()/RAND_MAX < mutation_rate){
        int row = rand()%n;
        ind.genes[row] = rand()%n;
        ind.fitness = fitness(ind.genes);
    }
}

// Selection: tournament
Individual selectParent() {
    int a=rand()%population_size;
    int b=rand()%population_size;
```

```
    return (population[a].fitness > population[b].fitness) ?
population[a] : population[b];
}

int main() {
    srand(time(0));
    // Initialize population
    for(int i=0;i<population_size;i++)
population.push_back(randomIndividual());

    int generation = 0;
    while(generation < max_generations){
        sort(population.begin(), population.end(), [](Individual
&a, Individual &b){ return a.fitness>b.fitness; });
        if(population[0].fitness == n*(n-1)/2) break; // solution
found

        vector<Individual> new_pop;
        while(new_pop.size()<population_size){
            Individual p1 = selectParent();
            Individual p2 = selectParent();
            Individual child = crossover(p1,p2);
            mutate(child);
            new_pop.push_back(child);
        }
        population = new_pop;
        generation++;
```

```
    }

    // Print solution
    cout<<"Solution found in generation
"<<generation<<"\n";
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(j==population[0].genes[i]) cout<<"Q ";
            else cout<<". ";
        }
        cout<<endl;
    }
}
```

Csp

```
#include<bits/stdc++.h>
using namespace std;
map<string,vector<string>>adj;
vector<string>clr;
map<string,string>ans;
bool ok(string u,string c){
    for(auto v:adj[u])
        if(ans[v]==c)return false;
    return true;
}
bool go(vector<string>&ct,int i){
    if(i==ct.size())return true;
    string u=ct[i];
    for(auto c:clr){
        if(ok(u,c)){
```

```
            ans[u]=c;
            if(go(ct,i+1))return true;
            ans[u]="";
        }
    }
    return false;
}
int main(){
    int n;
    cin>>n;
    vector<string>ct(n);
    for(int i=0;i<n;i++)cin>>ct[i];
    for(int i=0;i<n;i++){
        string u;
        cin>>u;
        string v;
        while(cin>>v&&v!="-1")
            adj[u].push_back(v);
    }
    int m;
    cin>>m;
    for(int i=0;i<m;i++){
        string c;
        cin>>c;
        clr.push_back(c);
    }

    if(go(ct,0)){
        for(auto u:ct)cout<<u<< " "<<ans[u]<<endl;
        map<string,int>cnt;
        for(auto u:ct)cnt[ans[u]]++;
        for(auto& x:cnt)cout<<x.first<<": "<<x.second<<endl;
    } else cout<<"No\n";
    return 0;
}
```