USING BFS]

#include <bits/stdc++.h>

using namespace std;

int x, y;

void printBoard(vector<vector<int>> temp)

{

    int n = temp.size();

    int m = temp[0].size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < m; j++)

        {

            cout << temp[i][j] << " ";

        }

        cout << endl;

    }

    cout << endl;

}

void findZero(vector<vector<int>> curr, int &x, int &y)

{

    int n = curr.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (curr[i][j] == 0)

            {

                x = i;

                y = j;

                break;

            }

        }

    }

}

void bfs(vector<vector<int>> &initial, vector<vector<int>> &goal)

{

    queue<vector<vector<int>>> q;

    int n = initial.size();

    set<vector<vector<int>>> vis;

    q.push(initial);

    vis.insert(initial);

    int delRow[] = {0, -1, 0, +1};

    int delCol[] = {-1, 0, +1, 0};

    while (!q.empty())

    {

        auto curr = q.front();

        q.pop();

        printBoard(curr);

        if (curr == goal)

        {

            cout << "Goal State found";

            break;

        }

        int x, y;

        findZero(curr, x, y);

        for (int i = 0; i < 4; i++)

        {

            int newx = x + delRow[i];

            int newy = y + delCol[i];

            if (newx >= 0 && newx < n && newy >= 0 && newy < n)

            {

                swap(curr[newx][newy], curr[x][y]);

                if (vis.find(curr) == vis.end())

                    q.push(curr);

                swap(curr[newx][newy], curr[x][y]);

            }

        }

    }

}

int main()

{

    vector<vector<int>> initial = {{2, 8, 3},

                                   {1, 6, 4},

                                   {7, 5, 0}};

    vector<vector<int>> goal = {{1, 2, 3},

                                {8, 0, 4},

                                {7, 6, 5}};

    bfs(initial, goal);

}

Using DFS]

#include <bits/stdc++.h>

using namespace std;

int x, y, flag;

void printBoard(vector<vector<int>> &temp)

{

    int n = temp.size();

    int m = temp[0].size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < m; j++)

        {

            cout << temp[i][j] << " ";

        }

        cout << endl;

    }

    cout << endl;

}

void findZero(vector<vector<int>> curr, int &x, int &y)

{

    int n = curr.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (curr[i][j] == 0)

            {

                x = i;

                y = j;

                return;

            }

        }

    }

}

int dfs(vector<vector<int>> &initial, vector<vector<int>> &goal, set<vector<vector<int>>> &vis)

{

    printBoard(initial);

    if (initial == goal)

    {

        cout << "Goal State found";

        return 1;

    }

    int n = initial.size();

    // q.push(initial);

    vis.insert(initial);

    int delRow[] = {0, -1, 0, +1};

    int delCol[] = {-1, 0, +1, 0};

    int x, y;

    findZero(initial, x, y);

    for (int i = 0; i < 4; i++)

    {

        int newx = x + delRow[i];

        int newy = y + delCol[i];

        if (newx >= 0 && newx < n && newy >= 0 && newy < n)

        {

            swap(initial[newx][newy], initial[x][y]);

            if (vis.find(initial) == vis.end())

                if (dfs(initial, goal, vis) == 1)

                    return 1;

            swap(initial[newx][newy], initial[x][y]);

        }

    }

}

int main()

{

    vector<vector<int>> initial = {{2, 8, 3},

                                   {1, 6, 4},

                                   {7, 5, 0}};

    vector<vector<int>> goal = {{0, 8, 3},

                                {2, 6, 4},

                                {1, 7, 5}};

    set<vector<vector<int>>> vis;

    dfs(initial, goal, vis);

}

3]Using DLS

#include <bits/stdc++.h>

using namespace std;

int x, y, flag;

void printBoard(vector<vector<int>> &temp)

{

    int n = temp.size();

    int m = temp[0].size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < m; j++)

        {

            cout << temp[i][j] << " ";

        }

        cout << endl;

    }

    cout << endl;

}

void findZero(vector<vector<int>> curr, int &x, int &y)

{

    int n = curr.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (curr[i][j] == 0)

            {

                x = i;

                y = j;

                return;

            }

        }

    }

}

int dfs(vector<vector<int>> &initial, vector<vector<int>> &goal, set<vector<vector<int>>> &vis, int depth, int curr\_depth)

{

    printBoard(initial);

    if (initial == goal && curr\_depth <= depth)

    {

        cout << "Goal State found";

        flag = 1;

        return 1;

    }

    if (curr\_depth > depth)

        return 0;

    int n = initial.size();

    // q.push(initial);

    vis.insert(initial);

    int delRow[] = {0, -1, 0, +1};

    int delCol[] = {-1, 0, +1, 0};

    int x, y;

    findZero(initial, x, y);

    for (int i = 0; i < 4; i++)

    {

        int newx = x + delRow[i];

        int newy = y + delCol[i];

        if (newx >= 0 && newx < n && newy >= 0 && newy < n)

        {

            swap(initial[newx][newy], initial[x][y]);

            if (vis.find(initial) == vis.end())

                if (dfs(initial, goal, vis, depth, curr\_depth + 1) == 1)

                    return 1;

            swap(initial[newx][newy], initial[x][y]);

        }

    }

}

int main()

{

    vector<vector<int>> initial = {{2, 8, 3},

                                   {1, 6, 4},

                                   {7, 5, 0}};

    vector<vector<int>> goal = {{0, 8, 3},

                                {2, 6, 4},

                                {1, 7, 5}};

    set<vector<vector<int>>> vis;

    int depth = 3;

    dfs(initial, goal, vis, depth, 0);

    if (!flag)

    {

        cout << "Goal state not exist with in limit";

    }

}

4]Depth iterative search

#include <bits/stdc++.h>

using namespace std;

class Node3

{

public:

    vector<vector<int>> board;

    int x, y;

    Node3 \*parent;

    int depth;

    Node3(vector<vector<int>> b, int xPos, int yPos, Node3 \*p, int d)

        : board(b), x(xPos), y(yPos), parent(p), depth(d) {}

};

vector<Node3 \*> getNeighbours(Node3 \*node)

{

    vector<Node3 \*> list;

    int row\_x[] = {1, 0, -1, 0};

    int row\_y[] = {0, -1, 0, 1};

    for (int i = 0; i < 4; i++)

    {

        int newx = node->x + row\_x[i];

        int newy = node->y + row\_y[i];

        if (newx >= 0 && newy >= 0 && newx < node->board.size() && newy < node->board[0].size())

        {

            vector<vector<int>> newboard = node->board;

            swap(newboard[node->x][node->y], newboard[newx][newy]);

            Node3 \*neighbour = new Node3(newboard, newx, newy, node, node->depth + 1);

            list.push\_back(neighbour);

        }

    }

    return list;

}

void printBoard(const vector<vector<int>> &board)

{

    for (const auto &row : board)

    {

        for (int val : row)

        {

            cout << val << " ";

        }

        cout << endl;

    }

    cout << endl;

}

void printPath(Node3 \*node)

{

    if (node == nullptr)

        return;

    printPath(node->parent);

    printBoard(node->board);

}

string serialize(const vector<vector<int>> &board)

{

    stringstream ss;

    for (const auto &row : board)

    {

        for (int val : row)

        {

            ss << val << ",";

        }

    }

    return ss.str();

}

bool depthLimitedDFS(Node3 \*node, set<string> &visited, const vector<vector<int>> &goal, int depthLimit)

{

    if (node->board == goal)

    {

        printPath(node);

        cout << "Goal state found" << endl;

        return true;

    }

    if (node->depth >= depthLimit)

        return false;

    visited.insert(serialize(node->board));

    for (Node3 \*neighbour : getNeighbours(node))

    {

        if (visited.find(serialize(neighbour->board)) == visited.end())

        {

            if (depthLimitedDFS(neighbour, visited, goal, depthLimit))

            {

                return true;

            }

        }

    }

    return false;

}

bool iterativeDeepeningDFS(const vector<vector<int>> &current, const vector<vector<int>> &goal, int maxDepth)

{

    int x = 0, y = 0;

    for (int i = 0; i < current.size(); i++)

    {

        for (int j = 0; j < current[i].size(); j++)

        {

            if (current[i][j] == 0)

            {

                x = i;

                y = j;

            }

        }

    }

    for (int depth = 0; depth <= maxDepth; depth++)

    {

        set<string> visited;

        Node3 \*start = new Node3(current, x, y, nullptr, 0);

        if (depthLimitedDFS(start, visited, goal, depth))

        {

            return true;

        }

    }

    return false;

}

int main()

{

    vector<vector<int>> initial = {

        {2, 8, 3},

        {1, 6, 4},

        {7, 0, 5}};

    vector<vector<int>> goal = {

        {1, 2, 3},

        {8, 0, 4},

        {7, 6, 5}};

    int maxDepth = 20; // You can adjust the max depth as needed

    if (!iterativeDeepeningDFS(initial, goal, maxDepth))

    {

        cout << "No solution found within depth limit " << maxDepth << endl;

    }

    return 0;

}

5]Hill Climbing algorithm

#include <bits/stdc++.h>

using namespace std;

void findzero(vector<vector<int>> &v, int &x, int &y)

{

    for (int i = 0; i < v.size(); i++)

    {

        for (int j = 0; j < v.size(); j++)

        {

            if (v[i][j] == 0)

            {

                x = i;

                y = j;

            }

        }

    }

}

void printvec(vector<vector<int>> &v1)

{

    for (auto it : v1)

    {

        for (auto i : it)

        {

            cout << i << " ";

        }

        cout << endl;

    }

}

int mismatch(vector<vector<int>> &v1, vector<vector<int>> &v2)

{

    int cnt = 0;

    int n = v1.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (v1[i][j] != v2[i][j])

                cnt++;

        }

    }

    return cnt;

}

void hillclimb(vector<vector<int>> &v1, vector<vector<int>> &v2)

{

    int n = v1.size();

    printvec(v1);

    int ini = mismatch(v1, v2);

    if (v1 == v2)

    {

        cout << "Goal State found";

        return;

    }

    int x, y;

    findzero(v1, x, y);

    int delR[] = {-1, 0, +1, 0};

    int delC[] = {0, -1, 0, +1};

    vector<pair<int, int>> st;

    for (int i = 0; i < 4; i++)

    {

        int newx = x + delR[i];

        int newy = y + delC[i];

        if (newx >= 0 && newx < n && newy >= 0 && newy < n)

        {

            swap(v1[newx][newy], v1[x][y]);

            int mis = mismatch(v1, v2);

            st.push\_back({mis, i});

            swap(v1[newx][newy], v1[x][y]);

        }

    }

    sort(st.begin(), st.end());

    int mini = st[0].first;

    int ops = st[0].second;

    if (mini < ini)

    {

        int newx = x + delR[ops];

        int newy = y + delC[ops];

        swap(v1[newx][newy], v1[x][y]);

        hillclimb(v1, v2);

    }

    else

    {

        return;

    }

}

int main()

{

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

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

    hillclimb(initial, final);

    return 0;

}

6]using A\* algorithm

#include <bits/stdc++.h>

using namespace std;

void printBoard(vector<vector<int>> temp)

{

    int n = temp.size();

    int m = temp[0].size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < m; j++)

        {

            cout << temp[i][j] << " ";

        }

        cout << endl;

    }

    cout << endl;

}

void findZero(vector<vector<int>> curr, int &x, int &y)

{

    int n = curr.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (curr[i][j] == 0)

            {

                x = i;

                y = j;

                break;

            }

        }

    }

}

int mismatch(vector<vector<int>> &curr, vector<vector<int>> &goal)

{

    int n = curr.size();

    int cnt = 0;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (curr[i][j] != goal[i][j])

                cnt++;

        }

    }

    return cnt;

}

void astar(vector<vector<int>> &initial, vector<vector<int>> &goal, int depth)

{

    priority\_queue<pair<pair<int, int>, vector<vector<int>>>, vector<pair<pair<int, int>, vector<vector<int>>>>, greater<pair<pair<int, int>, vector<vector<int>>>>> pq;

    pq.push({{mismatch(initial, goal), depth}, initial});

    int delx[] = {-1, 0, +1, 0};

    int dely[] = {0, -1, 0, +1};

    int n = initial.size();

    while (!pq.empty())

    {

        auto curr = pq.top().second;

        printBoard(curr);

        pq.pop();

        if (curr == goal)

        {

            cout << "Goal State found" << endl;

            return;

        }

        int currh = pq.top().first.second;

        int x, y;

        findZero(curr, x, y);

        for (int i = 0; i < 4; i++)

        {

            int newx = x + delx[i];

            int newy = y + dely[i];

            if (newx >= 0 && newx < n && newy >= 0 && newy < n)

            {

                swap(curr[newx][newy], curr[x][y]);

                int mi = mismatch(curr, goal);

                pq.push({{mi, currh + 1}, curr});

                swap(curr[newx][newy], curr[x][y]);

            }

        }

    }

}

int main()

{

    vector<vector<int>> initial = {{2, 8, 3},

                                   {1, 6, 4},

                                   {7, 5, 0}};

    vector<vector<int>> goal = {{1, 2, 3},

                                {8, 0, 4},

                                {7, 6, 5}};

    astar(initial, goal, 0);

    return 0;

}

7]AO \* algorithm

#include <iostream>

#include <map>

#include <vector>

#include <algorithm>

#include <sstream>

#include <string>

#include <numeric>

#include <climits>

using namespace std;

map<string, int> Cost(map<string, int> H, map<string, vector<string>> condition, int weight)

{

    map<string, int> cost;

    if (condition.find("AND") != condition.end())

    {

        vector<string> AND\_nodes = condition["AND"];

        string Path\_A;

        for (const auto &node : AND\_nodes)

        {

            if (!Path\_A.empty())

                Path\_A += " AND ";

            Path\_A += node;

        }

        int PathA = accumulate(AND\_nodes.begin(), AND\_nodes.end(), 0,

                               [&](int sum, const string &node)

                               {

                                   return sum + H[node] + weight;

                               });

        cost[Path\_A] = PathA;

    }

    if (condition.find("OR") != condition.end())

    {

        vector<string> OR\_nodes = condition["OR"];

        string Path\_B;

        for (const auto &node : OR\_nodes)

        {

            if (!Path\_B.empty())

                Path\_B += " OR ";

            Path\_B += node;

        }

        int PathB = INT\_MAX;

        for (const auto &node : OR\_nodes)

        {

            PathB = min(PathB, H[node] + weight);

        }

        cost[Path\_B] = PathB;

    }

    return cost; // j=1

}

map<string, map<string, int>> UpdateCost(

    map<string, int> H,

    map<string, map<string, vector<string>>> Conditions,

    int weight)

{

    vector<string> Main\_nodes;

    for (const auto &kv : Conditions)

    {

        Main\_nodes.push\_back(kv.first);

    }

    reverse(Main\_nodes.begin(), Main\_nodes.end());

    map<string, map<string, int>> least\_cost;

    for (const auto &key : Main\_nodes)

    {

        map<string, vector<string>> condition = Conditions[key];

        cout << key << ": ";

        for (const auto &kv : condition)

        {

            cout << kv.first << " "; // string and/or

            for (const auto &node : kv.second)

            {

                cout << node << " ";

            }

        }

        cout << ">>> ";

        map<string, int> c = Cost(H, condition, weight);

        for (const auto &kv : c)

        {

            cout << kv.first << " = " << kv.second << " ";

        }

        cout << endl;

        H[key] = min\_element(c.begin(), c.end(), [](const auto &a, const auto &b)

                             { return a.second < b.second; })

                     ->second;

        least\_cost[key] = Cost(H, condition, weight); // d:j=1

    }

    return least\_cost;

}

string ShortestPath(

    string Start,

    map<string, map<string, int>> Updated\_cost,

    map<string, int> H)

{

    string Path = Start;

    if (Updated\_cost.find(Start) != Updated\_cost.end())

    {

        int Min\_cost = min\_element(Updated\_cost[Start].begin(), Updated\_cost[Start].end(),

                                   [](const auto &a, const auto &b)

                                   { return a.second < b.second; })

                           ->second;

        vector<string> key;

        vector<int> values;

        for (const auto &kv : Updated\_cost[Start])

        {

            key.push\_back(kv.first);

            values.push\_back(kv.second);

        }

        auto it = find(values.begin(), values.end(), Min\_cost);

        int Index = distance(values.begin(), it);

        vector<string> Next;

        stringstream ss(key[Index]);

        string segment;

        while (getline(ss, segment, ' '))

        {

            if (segment != "AND" && segment != "OR")

            {

                Next.push\_back(segment);

            }

        }

        if (Next.size() == 1)

        {

            Start = Next[0];

            Path += "<--" + ShortestPath(Start, Updated\_cost, H);

        }

        else

        {

            Path += "<--(" + key[Index] + ") ";

            Start = Next[0];

            Path += "[" + ShortestPath(Start, Updated\_cost, H) + " + ";

            Start = Next[Next.size() - 1];

            Path += ShortestPath(Start, Updated\_cost, H) + "]";

        }

    }

    return Path;

}

int main()

{

    map<string, int> H;

    H["A"] = -1;

    H["B"] = 5;

    H["C"] = 2;

    H["D"] = 4;

    H["E"] = 7;

    H["F"] = 9;

    H["G"] = 3;

    H["H"] = 0;

    H["I"] = 0;

    H["J"] = 0;

    map<string, map<string, vector<string>>> Conditions;

    map<string, vector<string>> aConditions;

    aConditions["OR"] = {"B"};

    aConditions["AND"] = {"C", "D"};

    Conditions["A"] = aConditions;

    map<string, vector<string>> bConditions;

    bConditions["OR"] = {"E", "F"};

    Conditions["B"] = bConditions;

    map<string, vector<string>> cConditions;

    cConditions["OR"] = {"G"};

    cConditions["AND"] = {"H", "I"};

    Conditions["C"] = cConditions;

    map<string, vector<string>> dConditions;

    dConditions["OR"] = {"J"};

    Conditions["D"] = dConditions;

    // weight

    int weight = 1;

    // Updated cost

    cout << "Updated Cost :" << endl;

    map<string, map<string, int>> Updated\_cost = UpdateCost(H, Conditions, weight);

    cout << string(75, '\*') << endl;

    cout << "Shortest Path :" << endl;

    cout << ShortestPath("A", Updated\_cost, H) << endl;

    return 0;

}

8]CSP

#include <bits/stdc++.h>

using namespace std;

int possible(int r, int c, vector<string> &board)

{

    int dup\_r = r;

    int dup\_c = c;

    int n = board.size();

    while (r >= 0 && c >= 0)

    {

        if (board[r][c] == 'Q')

        {

            return 0;

        }

        r--;

        c--;

    }

    r = dup\_r;

    c = dup\_c;

    while (c >= 0)

    {

        if (board[r][c] == 'Q')

        {

            return 0;

        }

        c--;

    }

    r = dup\_r;

    c = dup\_c;

    while (r < n && c >= 0)

    {

        if (board[r][c] == 'Q')

        {

            return 0;

        }

        c--;

        r++;

    }

    return 1;

}

void nqueen(int col, vector<vector<string>> &ans, vector<string> &board)

{

    int n = board.size();

    if (col == n)

    {

        ans.push\_back(board);

        return;

    }

    for (int i = 0; i < n; i++)

    {

        if (possible(i, col, board))

        {

            board[i][col] = 'Q';

            nqueen(col + 1, ans, board);

            board[i][col] = '.';

        }

    }

}

int main()

{

    int n;

    cin >> n;

    vector<vector<string>> ans;

    vector<string> board(n);

    string s(n, '.');

    for (int i = 0; i < n; i++)

    {

        board[i] = s;

    }

    nqueen(0, ans, board);

    cout << "Number of configuration" << ans.size() << endl;

    for (auto it : ans)

    {

        for (auto i : it)

        {

            cout << i << endl;

        }

        cout << endl;

    }

    return 0;

}

9]min-max algorithm

import java.util.Scanner;

public class MinMaxAI {

    private char[][] board;

    private char humanPlayer;

    private char aiPlayer;

    private char currentPlayer;

    public MinMaxAI() {

        board = new char[3][3];

        humanPlayer = 'X';

        aiPlayer = 'O';

        currentPlayer = humanPlayer;

        initializeBoard();

    }

    private void initializeBoard() {

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                board[i][j] = '-';

            }

        }

    }

    private void printBoard() {

        System.out.println("-------------");

        for (int i = 0; i < 3; i++) {

            System.out.print("| ");

            for (int j = 0; j < 3; j++) {

                System.out.print(board[i][j] + " | ");

            }

            System.out.println();

            System.out.println("-------------");

        }

    }

    private boolean isBoardFull() {

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                if (board[i][j] == '-') {

                    return false;

                }

            }

        }

        return true;

    }

    private boolean isValidMove(int row, int col) {

        if(row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == '-'){

            return true;

        }else{

            System.out.println("Please enter valid move!.");

            return false;

        }

    }

    private void makeMove(int row, int col, char player) {

        board[row][col] = player;

    }

    private boolean checkForWinner(char player) {

        // Check rows, columns, and diagonals

        for (int i = 0; i < 3; i++) {

            if (board[i][0] == player && board[i][1] == player && board[i][2] == player) {

                return true; // Row win

            }

            if (board[0][i] == player && board[1][i] == player && board[2][i] == player) {

                return true; // Column win

            }

        }

        if (board[0][0] == player && board[1][1] == player && board[2][2] == player) {

            return true; // Diagonal win (top-left to bottom-right)

        }

        if (board[0][2] == player && board[1][1] == player && board[2][0] == player) {

            return true; // Diagonal win (top-right to bottom-left)

        }

        return false;

    }

    private int minimax(int depth, boolean isMaximizing) {

        if (checkForWinner(humanPlayer)) {

            return depth - 10; // Human wins, penalize depth

        } else if (checkForWinner(aiPlayer)) {

            return 10 - depth; // AI wins, reward depth

        } else if (isBoardFull()) {

            return 0; // Draw

        }

        if (isMaximizing) {

            int maxEval = Integer.MIN\_VALUE;

            for (int i = 0; i < 3; i++) {

                for (int j = 0; j < 3; j++) {

                    if (board[i][j] == '-') {

                        board[i][j] = aiPlayer;

                        maxEval = Math.max(maxEval, minimax(depth + 1, false));

                        board[i][j] = '-';

                    }

                }

            }

            return maxEval;

        } else {

            int minEval = Integer.MAX\_VALUE;

            for (int i = 0; i < 3; i++) {

                for (int j = 0; j < 3; j++) {

                    if (board[i][j] == '-') {

                        board[i][j] = humanPlayer;

                        minEval = Math.min(minEval, minimax(depth + 1, true));

                        board[i][j] = '-';

                    }

                }

            }

            return minEval;

        }

    }

    private int[] getBestMove() {

        int[] bestMove = new int[]{-1, -1};

        int bestScore = Integer.MIN\_VALUE;

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                if (board[i][j] == '-') {

                    board[i][j] = aiPlayer;

                    int score = minimax(0, false);

                    board[i][j] = '-';

                    if (score > bestScore) {

                        bestScore = score;

                        bestMove[0] = i;

                        bestMove[1] = j;

                    }

                }

            }

        }

        return bestMove;

    }

    public void play() {

        Scanner scanner = new Scanner(System.in);

        int row, col;

        do {

            printBoard();

            if (currentPlayer == humanPlayer) {

                do {

                    System.out.println("Player " + humanPlayer + ", enter your move (row and column): ");

                    row = scanner.nextInt() - 1;

                    col = scanner.nextInt() - 1;

                } while (!isValidMove(row, col));

                makeMove(row, col, humanPlayer);

            } else {

                int[] aiMove = getBestMove();

                makeMove(aiMove[0], aiMove[1], aiPlayer);

            }

            switchPlayer();

        } while (!checkForWinner(humanPlayer) && !checkForWinner(aiPlayer) && !isBoardFull());

        printBoard();

        if (checkForWinner(humanPlayer)) {

            System.out.println("Human wins!");

        } else if (checkForWinner(aiPlayer)) {

            System.out.println("AI wins!");

        } else {

            System.out.println("It's a draw!");

        }

        scanner.close();

    }

    private void switchPlayer() {

        if (currentPlayer == humanPlayer) {

            currentPlayer = aiPlayer;

        } else {

            currentPlayer = humanPlayer;

        }

    }

    public static void main(String[] args) {

        MinMaxAI game = new MinMaxAI();

        game.play();

    }

}

10[Non AI hame

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Set up the game board as an array

vector<string> board = {"-", "-", "-", "-", "-", "-", "-", "-", "-"};

// Define a function to print the game board

void printBoard()

{

    cout << board[0] << " | " << board[1] << " | " << board[2] << endl;

    cout << board[3] << " | " << board[4] << " | " << board[5] << endl;

    cout << board[6] << " | " << board[7] << " | " << board[8] << endl;

}

// Define a function to handle a player's turn

void takeTurn(string player)

{

    cout << player << "'s turn." << endl;

    cout << "Choose a position from 1-9: ";

    int position;

    cin >> position;

    position -= 1;

    while (position < 0 || position > 8 || board[position] != "-")

    {

        cout << "Invalid input or position already taken. Choose a different position: ";

        cin >> position;

        position -= 1;

    }

    board[position] = player;

    printBoard();

}

// Define a function to check if the game is over

string checkGameOver()

{

    // Check for a win

    if ((board[0] == board[1] && board[1] == board[2] && board[0] != "-") ||

        (board[3] == board[4] && board[4] == board[5] && board[3] != "-") ||

        (board[6] == board[7] && board[7] == board[8] && board[6] != "-") ||

        (board[0] == board[3] && board[3] == board[6] && board[0] != "-") ||

        (board[1] == board[4] && board[4] == board[7] && board[1] != "-") ||

        (board[2] == board[5] && board[5] == board[8] && board[2] != "-") ||

        (board[0] == board[4] && board[4] == board[8] && board[0] != "-") ||

        (board[2] == board[4] && board[4] == board[6] && board[2] != "-"))

    {

        return "win";

    }

    // Check for a tie

    else if (count(board.begin(), board.end(), "-") == 0)

    {

        return "tie";

    }

    // Game is not over

    else

    {

        return "play";

    }

}

// Define the main game loop

int main()

{

    printBoard();

    string currentPlayer = "X";

    bool gameOver = false;

    while (!gameOver)

    {

        takeTurn(currentPlayer);

        string gameResult = checkGameOver();

        if (gameResult == "win")

        {

            cout << currentPlayer << " wins!" << endl;

            gameOver = true;

        }

        else if (gameResult == "tie")

        {

            cout << "It's a tie!" << endl;

            gameOver = true;

        }

        else

        {

            // Switch to the other player

            currentPlayer = currentPlayer == "X" ? "O" : "X";

        }

    }

    return 0;

}

11]Prolog

% Facts

parent(john, mary).

parent(john, mike).

parent(susan, mary).

parent(susan, mike).

parent(mary, alice).

parent(mary, bob).

parent(mike, james).

parent(mike, lily).

male(john).

male(mike).

male(bob).

male(james).

female(susan).

female(mary).

female(alice).

female(lily).

% Rules

% A is a father of B if A is a parent of B and A is male

father(A, B) :-

parent(A, B),

male(A).

% A is a mother of B if A is a parent of B and A is female

mother(A, B) :-

parent(A, B),

female(A).

% A is a grandparent of B if A is a parent of C and C is a parent of B

grandparent(A, B) :-

parent(A, C),

parent(C, B).

% A is a grandchild of B if B is a grandparent of A

grandchild(A, B) :-

grandparent(B, A).

% A is a sibling of B if they share at least one parent

sibling(A, B) :-

parent(C, A),

parent(C, B),

A \= B.

% A is a brother of B if A is a sibling of B and A is male

brother(A, B) :-

sibling(A, B),

male(A).

% A is a sister of B if A is a sibling of B and A is female

sister(A, B) :-

sibling(A, B),

female(A).

% Who are Mary's parents?

?- parent(X, mary).

% Expected output: X = john ; X = susan.

% Who are John's grandchildren?

?- grandchild(X, john).

% Expected output: X = alice ; X = bob ; X = james ; X = lily.

% Who are Mary's siblings?

?- sibling(X, mary).

% Expected output: X = mike.

% Who are Mike's children?

?- parent(mike, X).

% Expected output: X = james ; X = lily.

% Who is James' grandfather?

?- grandparent(X, james), male(X).

% Expected output: X = john.

% Who are Bob's uncles?

?- brother(X, mary), parent(mary, bob).

% Expected output: X = mike.