**Btech CSE 4th Semester**

**Feb – June 2022**

**AI & Application Lab Exercises**

**Course Code : 14B17CI772**



**Department of Computer Science and Engineering**

**Jaypee University**

**of Engineering and Technology, Guna**

**Name: Palash Mishra**

**Enroll: 201B172**

**Course Description**

**Title of Course: AI & Application Lab Course Code:** 14B17CI772

**L-T-P Scheme: 0-0-2** **Course Credit: 1**

**Objectives:** In this course we will implement the basic components of an intelligent system,

their functions, mechanisms, policies and techniques used in their implementation and examples.

**Learning Outcomes:** The students will have a detailed knowledge of the concepts of artificial

intelligence. Various applications of AI in different fields, aware of a variety of approaches to AI techniques.

**Course Contents:**

**Unit-1**: Introduction to AI and intelligent agents

**Unit-2:** Problem solving, Problem spaces and blind search techniques, informed search techniques,

Constraint satisfaction problems

**Unit-3:** Knowledge representation and reasoning techniques, Logic programming, Logical agents,

Game playing, Planning,

**Unit-4:** Learning, Reasoning under uncertain situations,

**Unit-5:** Expert systems, Decision support systems, Domain specific AI applications.

**List of AI Problems for Lab-**

**Problem 1: Write a program to solve “Tower of Hanoi” with only 2 disks and then 3 disks.**

**Problem 2: Write a program to solve “4-Queens” and “8-Queens” puzzle.**

**Problem 3: Write a program to solve “4-color map” problem.**

**Problem 4: Write a program to solve “8 – puzzle” and “15-puzzle” take any initial and goal state.**

**Problem 5: Write a program to solve “Latin Square” problem.**

**Problem 6: Code the game: Tick-Tack-Toe.**

**Problem 7: Code the game: Checkers.**

**Problem 8: Write a program to solve water jug problem.**

“You are given an m liter jug and a n liter jug. Both the jugs are initially empty. The jugs don’t

have markings to allow measuring smaller quantities. You have to use the jugs to measure

d liters of water where d is less than n.”

**Problem 9: Write a Program to Implement Missionaries-Cannibals Problems.**

“In this problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, that the missionaries present on the bank cannot be outnumbered by cannibals. The boat cannot cross the river by itself with no people on board.”

**Problem 10: Write a program to solve traveling salesman problem.**

“Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.”

**Problem 11: Write a Program to Implement Camel and Banana Puzzle.**

“A person has 3000 bananas and a camel. The person wants to transport the maximum number of bananas to a destination which is 1000 KMs away, using only the camel as a mode of transportation. The camel cannot carry more than 1000 bananas at a time and eats a banana every km it travels. What is the maximum number of bananas that can be transferred to the destination using only camel (no other mode of transportation is allowed).”

**Problem 12: Write a program of Magic Square.**

“A magic square of order n is an arrangement of n2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant.”

**Problem 13: Knight’s Tour problem using Backtracking.**

“Given an N\*N board with the Knight placed on the first block of an empty board. Moving according to the rules of chess knight must visit each square exactly once. Print the order of each cell in which they are visited.”

**Problem 14: Code the game software: Chess.**

**Problem 15: K-Means Algorithm for Colour Compression.**

“One interesting application of clustering is in colour compression within images. For example, imagine you have an image with millions of colours. In most images, a large number of the colours will be unused, and many of the pixels in the image will have similar or even identical colours.”

**Text Books**

1. Rich, Elaine Knight, Kevin, Artificial Intelligence, Tata McGraw Hill.
2. Luger, George F, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Pearson Education.

**References**

1. Nilsson, Nils J, Artificial Intelligence, Morgan Kaufmann
2. Russell, Stuart J. Norvig, Peter, Artificial Intelligence: A Modern Approach,
3. Pearson Education
4. Negnevitsky, Michael, Artificial Intelligence: A Guide to Intelligent Systems, Addison-Wesley.

**# Tower of Hanoi**

include<stdio.h>

#include<conio.h>

void tower(int, char, char, char);

int count=0;

int main()

{ int num;

printf("Enter the number of disks : ");

scanf("%d", &num);

printf("The sequence of moves involved in the Tower of Hanoi are :\n");

tower(num, 'A', 'C', 'B');

printf("\nThe total number of moves involved:%d",count);

getch();

return 0;

}

void tower(int num, char frompeg, char topeg, char inpeg)

{ if (num == 1)

{

count++;

printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);

return;

}

tower(num - 1, frompeg, inpeg, topeg);

count++;

printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);

tower(num - 1, inpeg, topeg, frompeg);

**# N-Queens**

include<stdio.h>

#include<conio.h>

#include<math.h>

int a[30],count=0;

int place(int pos)

{ int i;

for(i=1;i<pos;i++)

{

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))

return 0;

}

return 1;

}

void print\_sol(int n)

{

int i,j;

count++;

printf("\n\nSolution #%d:\n",count);

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(a[i]==j)

printf("Q\t");

else

printf("\*\t");

}

printf("\n");

}

}

void queen(int n)

{

int k=1;

a[k]=0;

while(k!=0)

{

a[k]=a[k]+1;

while((a[k]<=n)&&!place(k))

a[k]++;

if(a[k]<=n)

{

if(k==n)

print\_sol(n);

else

{

k++;

a[k]=0;

}

}

else

k--; }}

void main()

{

int i,n;

clrscr();

printf("Enter the number of Queens\n");

scanf("%d",&n);

queen(n);

printf("\nTotal solutions=%d",count);

getch();

**# m-Colouring**

#include<bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 4

void printSolution(int color[]);

// check if the colored

// graph is safe or not

bool isSafe(bool graph[V][V], int color[])

{

// check for every edge

for (int i = 0; i < V; i++)

for (int j = i + 1; j < V; j++)

if (graph[i][j] && color[j] == color[i])

return false;

return true;

}

bool graphColoring(bool graph[V][V], int m, int i,

int color[V])

{

// if current index reached end

if (i == V) {

// if coloring is safe

if (isSafe(graph, color)) {

// Print the solution

printSolution(color);

return true;

}

return false;

}

// Assign each color from 1 to m

for (int j = 1; j <= m; j++) {

color[i] = j;

// Recur of the rest vertices

if (graphColoring(graph, m, i + 1, color))

return true;

color[i] = 0;

}

return false;

}

/\* A utility function to print solution \*/

void printSolution(int color[])

{

cout << "Solution Exists:" " Following are the assigned colors \n";

for (int i = 0; i < V; i++)

cout << " " << color[i];

cout << "\n";

}

// Driver code

int main()

{

bool graph[V][V] = {

{ 0, 1, 1, 1 },

{ 1, 0, 1, 0 },

{ 1, 1, 0, 1 },

{ 1, 0, 1, 0 },

};

int m = 3; // Number of colors

// Initialize all color values as 0.

// This initialization is needed

// correct functioning of isSafe()

int color[V];

for (int i = 0; i < V; i++)

color[i] = 0;

if (!graphColoring(graph, m, 0, color))

cout << "Solution does not exist";

return 0;

}

**# 8 Puzzle**

// Program to print path from root node to destination node

// for N\*N -1 puzzle algorithm using Branch and Bound

// The solution assumes that instance of puzzle is solvable

#include <bits/stdc++.h>

using namespace std;

#define N 3

// state space tree nodes

struct Node

{

// stores the parent node of the current node

// helps in tracing path when the answer is found

Node\* parent;

// stores matrix

int mat[N][N];

// stores blank tile coordinates

int x, y;

// stores the number of misplaced tiles

int cost;

// stores the number of moves so far

int level;

};

// Function to print N x N matrix

int printMatrix(int mat[N][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

printf("%d ", mat[i][j]);

printf("\n");

}

}

// Function to allocate a new node

Node\* newNode(int mat[N][N], int x, int y, int newX,

int newY, int level, Node\* parent)

{

Node\* node = new Node;

// set pointer for path to root

node->parent = parent;

// copy data from parent node to current node

memcpy(node->mat, mat, sizeof node->mat);

// move tile by 1 position

swap(node->mat[x][y], node->mat[newX][newY]);

// set number of misplaced tiles

node->cost = INT\_MAX;

// set number of moves so far

node->level = level;

// update new blank tile coordinates

node->x = newX;

node->y = newY;

return node;

}

// bottom, left, top, right

int row[] = { 1, 0, -1, 0 };

int col[] = { 0, -1, 0, 1 };

// Function to calculate the number of misplaced tiles

// ie. number of non-blank tiles not in their goal position

int calculateCost(int initial[N][N], int final[N][N])

{

int count = 0;

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

if (initial[i][j] && initial[i][j] != final[i][j])

count++;

return count;

}

// Function to check if (x, y) is a valid matrix coordinate

int isSafe(int x, int y)

{

return (x >= 0 && x < N && y >= 0 && y < N);

}

// print path from root node to destination node

void printPath(Node\* root)

{

if (root == NULL)

return;

printPath(root->parent);

printMatrix(root->mat);

printf("\n");

}

// Comparison object to be used to order the heap

struct comp

{

bool operator()(const Node\* lhs, const Node\* rhs) const

{

return (lhs->cost + lhs->level) > (rhs->cost + rhs->level);

}

};

// Function to solve N\*N - 1 puzzle algorithm using

// Branch and Bound. x and y are blank tile coordinates

// in initial state

void solve(int initial[N][N], int x, int y,

int final[N][N])

{

// Create a priority queue to store live nodes of

// search tree;

priority\_queue<Node\*, std::vector<Node\*>, comp> pq;

// create a root node and calculate its cost

Node\* root = newNode(initial, x, y, x, y, 0, NULL);

root->cost = calculateCost(initial, final);

// Add root to list of live nodes;

pq.push(root);

// Finds a live node with least cost,

// add its childrens to list of live nodes and

// finally deletes it from the list.

while (!pq.empty())

{

// Find a live node with least estimated cost

Node\* min = pq.top();

// The found node is deleted from the list of

// live nodes

pq.pop();

// if min is an answer node

if (min->cost == 0)

{

// print the path from root to destination;

printPath(min);

return;

}

// do for each child of min

// max 4 children for a node

for (int i = 0; i < 4; i++)

{

if (isSafe(min->x + row[i], min->y + col[i]))

{

// create a child node and calculate

// its cost

Node\* child = newNode(min->mat, min->x,

min->y, min->x + row[i],

min->y + col[i],

min->level + 1, min);

child->cost = calculateCost(child->mat, final);

// Add child to list of live nodes

pq.push(child);

}

}

}

}

// Driver code

int main()

{

// Initial configuration

// Value 0 is used for empty space

int initial[N][N] =

{

{1, 2, 3},

{5, 6, 0},

{7, 8, 4}

};

// Solvable Final configuration

// Value 0 is used for empty space

int final[N][N] =

{

{1, 2, 3},

{5, 8, 6},

{0, 7, 4}

};

// Blank tile coordinates in initial

// configuration

int x = 1, y = 2;

solve(initial, x, y, final);

return 0;

}

**# Latin Square**

#include<stdio.h>

// Function to print n x n Latin Square

void printLatin(int n)

{

// A variable to control the rotation

// point.

int k = n+1;

// Loop to print rows

for (int i=1; i<=n; i++)

{

// This loops runs only after first

// iteration of outer loop. It prints

// numbers from n to k

int temp = k;

while (temp <= n)

{

printf("%d ", temp);

temp++;

}

// This loop prints numbers from 1 to k-1.

for (int j=1; j<k; j++)

printf("%d ", j);

k--;

printf("\n");

}

}

// Driver program to test above function

int main(void)

{

int n = 5;

// Invoking printLatin function

printLatin(n);

return 0;

}

**# Tic Tac Toe**

// A C++ Program to play tic-tac-toe

#include<bits/stdc++.h>

using namespace std;

#define COMPUTER 1

#define HUMAN 2

#define SIDE 3 // Length of the board

// Computer will move with 'O'

// and human with 'X'

#define COMPUTERMOVE 'O'

#define HUMANMOVE 'X'

// A function to show the current board status

void showBoard(char board[][SIDE])

{

printf("\n\n");

printf("\t\t\t %c | %c | %c \n", board[0][0],

board[0][1], board[0][2]);

printf("\t\t\t--------------\n");

printf("\t\t\t %c | %c | %c \n", board[1][0],

board[1][1], board[1][2]);

printf("\t\t\t--------------\n");

printf("\t\t\t %c | %c | %c \n\n", board[2][0],

board[2][1], board[2][2]);

return;

}

// A function to show the instructions

void showInstructions()

{

printf("\t\t\t Tic-Tac-Toe\n\n");

printf("Choose a cell numbered from 1 to 9 as below"

" and play\n\n");

printf("\t\t\t 1 | 2 | 3 \n");

printf("\t\t\t--------------\n");

printf("\t\t\t 4 | 5 | 6 \n");

printf("\t\t\t--------------\n");

printf("\t\t\t 7 | 8 | 9 \n\n");

printf("-\t-\t-\t-\t-\t-\t-\t-\t-\t-\n\n");

return;

}

// A function to initialise the game

void initialise(char board[][SIDE], int moves[])

{

// Initiate the random number generator so that

// the same configuration doesn't arises

srand(time(NULL));

// Initially the board is empty

for (int i=0; i<SIDE; i++)

{

for (int j=0; j<SIDE; j++)

board[i][j] = ' ';

}

// Fill the moves with numbers

for (int i=0; i<SIDE\*SIDE; i++)

moves[i] = i;

// randomise the moves

random\_shuffle(moves, moves + SIDE\*SIDE);

return;

}

// A function to declare the winner of the game

void declareWinner(int whoseTurn)

{

if (whoseTurn == COMPUTER)

printf("COMPUTER has won\n");

else

printf("HUMAN has won\n");

return;

}

// A function that returns true if any of the row

// is crossed with the same player's move

bool rowCrossed(char board[][SIDE])

{

for (int i=0; i<SIDE; i++)

{

if (board[i][0] == board[i][1] &&

board[i][1] == board[i][2] &&

board[i][0] != ' ')

return (true);

}

return(false);

}

// A function that returns true if any of the column

// is crossed with the same player's move

bool columnCrossed(char board[][SIDE])

{

for (int i=0; i<SIDE; i++)

{

if (board[0][i] == board[1][i] &&

board[1][i] == board[2][i] &&

board[0][i] != ' ')

return (true);

}

return(false);

}

// A function that returns true if any of the diagonal

// is crossed with the same player's move

bool diagonalCrossed(char board[][SIDE])

{

if (board[0][0] == board[1][1] &&

board[1][1] == board[2][2] &&

board[0][0] != ' ')

return(true);

if (board[0][2] == board[1][1] &&

board[1][1] == board[2][0] &&

board[0][2] != ' ')

return(true);

return(false);

}

// A function that returns true if the game is over

// else it returns a false

bool gameOver(char board[][SIDE])

{

return(rowCrossed(board) || columnCrossed(board)

|| diagonalCrossed(board) );

}

// A function to play Tic-Tac-Toe

void playTicTacToe(int whoseTurn)

{

// A 3\*3 Tic-Tac-Toe board for playing

char board[SIDE][SIDE];

int moves[SIDE\*SIDE];

// Initialise the game

initialise(board, moves);

// Show the instructions before playing

showInstructions();

int moveIndex = 0, x, y;

// Keep playing till the game is over or it is a draw

while (gameOver(board) == false &&

moveIndex != SIDE\*SIDE)

{

if (whoseTurn == COMPUTER)

{

x = moves[moveIndex] / SIDE;

y = moves[moveIndex] % SIDE;

board[x][y] = COMPUTERMOVE;

printf("COMPUTER has put a %c in cell %d\n",

COMPUTERMOVE, moves[moveIndex]+1);

showBoard(board);

moveIndex ++;

whoseTurn = HUMAN;

}

else if (whoseTurn == HUMAN)

{

x = moves[moveIndex] / SIDE;

y = moves[moveIndex] % SIDE;

board[x][y] = HUMANMOVE;

printf ("HUMAN has put a %c in cell %d\n",

HUMANMOVE, moves[moveIndex]+1);

showBoard(board);

moveIndex ++;

whoseTurn = COMPUTER;

}

}

// If the game has drawn

if (gameOver(board) == false &&

moveIndex == SIDE \* SIDE)

printf("It's a draw\n");

else

{

// Toggling the user to declare the actual

// winner

if (whoseTurn == COMPUTER)

whoseTurn = HUMAN;

else if (whoseTurn == HUMAN)

whoseTurn = COMPUTER;

// Declare the winner

declareWinner(whoseTurn);

}

return;

}

// Driver program

int main()

{

// Let us play the game with COMPUTER starting first

playTicTacToe(COMPUTER);

return (0);

}

**# Water Jug**

#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> pii;

void printpath(map<pii,pii>mp ,pii u)

{

if(u.first==0 &&u.second==0)

{

cout<<0<<" "<<0<<endl;

return ;

}

printpath(mp,mp[u]);

cout<<u.first<<" "<<u.second<<endl;

}

void BFS(int a ,int b, int target)

{

map<pii, int>m;

bool isSolvable =false;

vector<tuple<int ,int ,int>>path;

map<pii, pii>mp;

queue<pii>q;

q.push(make\_pair(0,0));

while(!q.empty())

{

auto u =q.front();

// cout<<u.first<<" "<<u.second<<endl;

q.pop();

if(m[u]==1)

continue;

if ((u.first > a || u.second > b || u.first < 0 || u.second < 0))

continue;

// cout<<u.first<<" "<<u.second<<endl;

m[{u.first,u.second}]=1;

if(u.first == target || u.second==target)

{

isSolvable = true;

printpath(mp,u);

if (u.first == target) {

if (u.second != 0)

cout<<u.first<<" "<<0<<endl;

}

else {

if (u.first != 0)

cout<<0<<" "<<u.second<<endl;

}

return;

}

// completely fill the jug 2

if(m[{u.first,b}]!=1)

{q.push({u.first,b});

mp[{u.first,b}]=u;}

// completely fill the jug 1

if(m[{a,u.second}]!=1)

{ q.push({a,u.second});

mp[{a,u.second}]=u;}

//transfer jug 1 -> jug 2

int d = b - u.second;

if(u.first >= d)

{

int c = u.first - d;

if(m[{c,b}]!=1)

{q.push({c,b});

mp[{c,b}]=u;}

}

else

{

int c = u.first + u.second;

if(m[{0,c}]!=1)

{q.push({0,c});

mp[{0,c}]=u;}

}

//transfer jug 2 -> jug 1

d = a - u.first;

if(u.second >= d)

{

int c = u.second - d;

if(m[{a,c}]!=1)

{q.push({a,c});

mp[{a,c}]=u;}

}

else

{

int c = u.first + u.second;

if(m[{c,0}]!=1)

{q.push({c,0});

mp[{c,0}]=u;}

}

// empty the jug 2

if(m[{u.first,0}]!=1)

{ q.push({u.first,0});

mp[{u.first,0}]=u;}

// empty the jug 1

if(m[{0,u.second}]!=1)

{q.push({0,u.second});

mp[{0,u.second}]=u;}

}

if (!isSolvable)

cout << "No solution";

}

int main()

{

int Jug1 = 4, Jug2 = 3, target = 2;

cout << "Path from initial state "

"to solution state ::\n";

BFS(Jug1, Jug2, target);

return 0;

}