classmate

Date
Page

yourself vs system

20/08/25 Week 1

1. Implement a Tic Tac Toe game -

Algorithm TicTacToe (board) {
// ~~Initialize~~ Print the board
for i in range (3)
~~print~~ for j in range (3)
print (board[i][j])

// Initialize the board
for ~~&~~i from 0 to 3
for j from 0 to 3
board[i][j]=='-'

// Take input from users
// Player 1 - X
// Player 2 - 0
print (~~Enter~~ Player 1 enter position where you want to place X)
Input
print (Player 2 enter position where you want to place 0)
win (board) // Check for winning conditions
check if place is empty

// Winning conditions check
for i from 0 to 3
if (board[0][i] != X // 0)
break;
if (board[i][0] != X/0)
break;

for i from 0 to 3
    for j from 0 to 3
        if (i == j)
            if (board[i][j] != 'X' || 'O')
                break;

Output:

Enter moves as 'row col'

|   | 0 | 1 | 2 |
|---|---|---|---|
| 0 | - | - | - |
| 1 | - | - | - |
| 2 | - | - | - |

Player X, enter your move : 0 1

|   | 0 | 1 | 2 |
|---|---|---|---|
| 0 | - | X | - |
| 1 | - | - | - |
| 2 | - | - | - |

Player O, your move: 1 1
Player X, your move: 0 0
Player O, your move: 0 0
Invalid move! Position is taken
Player O, your move: 2 2
Player X, your move: 0 2

|   | 0 | 1 | 2 |
|---|---|---|---|
| 0 | X | X | X |
| 1 |   | O |   |
| 2 |   |   | O |

Player X wins!
Do you want to play again? (y/n)
Thanks for playing

2) Implement vacuum cleaner

Clean = 0
dirty = 1

Algorithm Vacuum Cleaner (rooms)
for i from 0 to 3
for j from 0 to 3
if(rooms[i][j] == 'clean')
move to next room
else
clean and move to next room

move to next room:

| | |
|---|---|
| 00 | 01 |
| 10 | 11 |

```
j++;
if (j==1)
    i++;
```

clean and move to next room:

```
rooms[i][j] == 'clean'
j++;
if (j==1)
    i++;
```

2) Vacuum cleaner

Algorithm vacuumCleaner (rooms)
// Initialize all rooms to dirty
// clean = 0 dirty = 1
for i from 0 to ~~3~~ 1
for j from 0 to ~~3~~ 1
rooms[i][j] = 1

// Start cleaning
for i from 0 to 1
~~i~~for j from 0 to 1
if(rooms[i][j]== 1) → clean(rooms)
~~rooms[i][j]=0~~
j++;
if (j==1)
i++;
else
j++;
if(j==1)
i++;

def clean(rooms):
rooms[i][j]=0;

20/8/19

classmate
Date
Page

Output:
Initial room status: [0, 1, 0, 1]
Room 1 is already clean.
Moving to room 2.
Room 2 is Dirty. Cleaning...
Room 2 is now clean.
Moving to Room 3
Room 3 is already clean.
Room 4 is dirty. Cleaning
Room 4 is now clean.

2/1/8/25

Week 2

1. 8 puzzle - Misplaced tiles

```
function misplaced^Algo (start, goal):
    open = priority queue ordered by f = g + h
    closed = empty set // states already visited
    g(start) = 0
    h(start) = misplaced(start, goal)
    push(f, start, path = [start]) into open
    while open not empty:
        ~~f, st~~ (f, state, path) = pop lowest f from open
        if state == goal:
            return path
        add state to closed
        for neighbor in Expand(state):
            if neighbor not in closed:
                g(neighbor) = g(state) + 1
                h(neighbor) = misplaced(neighbor, goal)
                f(neighbor) = g(neighbor) + h(neighbor)
                push(f, neighbor, path + [neighbor]) into open
        return "No Solution"
function misplaced(state, goal):
    count = 0
    for i in 0 to 8:
        if state[i] != 0 and state[i] != goal[i]:
            count++
    return count
```

$$f(n) = \underbrace{g(n)}_{\text{cost from start to current state}} + \underbrace{h(n)}_{\text{misplaced tiles count}}$$

2. 8 puzzle - Manhattan

function ManhattanAlgo(start, goal):
    Put start in open with $f = g + h$
    $g = 0$ , $h = manhattan(start, goal)$

~~Output:~~

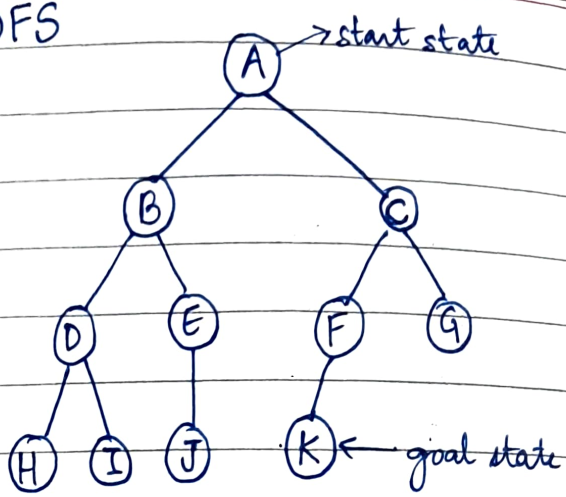
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | | 1 | 2 | 3 | | 1 | 2 | 3 |
| 4 | 5 | 6 | | 4 | 5 | 6 | | 4 | 5 | 6 |
| 7 | 8 | 0 | | 0 | 7 | 8 | | 7 | 8 | 0 |
| goal state | | | | start state | | | | $f=2$ | | |

Output:
Solution found in 2 moves
(1, 2, 3)
(4, 5, 6)
(0, 7, 8)

(1, 2, 3)
(4, 5, 6)
(7, 0, 8)

(1, 2, 3)
(4, 5, 6)
(7, 8, 0

2. IDDFS




A

A B C

A B D E C F G

A B D H I E J C F K G

IDDFS (start, goal, max_depth):
    for depth = 0 to max_depth
        visited = [ ]
        result = DFS(start, goal, depth, visited)
        if result == found
            return true
    return false

DFS( start, goal, depth, visited):
    if node == goal &
        return found
    ~~if limit == 0~~
        ~~return~~

```
mark node as visited
  for each neighbor of node:
    if neighbor not in visited:
      result = DFS(neighbor, goal, limit, -1, visited)
  if result == found
     return found
  return not found
```

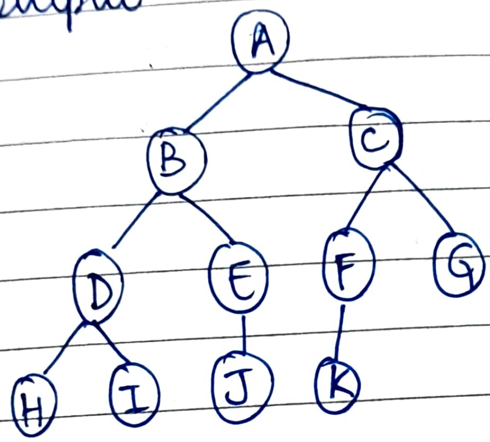
~~Output:~~

```
result = DFS(node.left, goal, limit-1)
if result == found
     return found
 result = DFS(node.right, goal, limit-1)
  if result == found
     return found
return not_found
```

```
max depth(node):
   if node is NULL:
      return 0
   left_depth = maxdepth(node.left)
   right_depth = maxdepth(node.right)
   return 1+ max(left_depth, right_depth)
```

~~3. 8 Puzzle Manhattan~~

Output :




goal K found within depth limit.

classmate

Date

Page

3. 8 Puzzle - Manhattan

```
function ManhattanAlgo (start, goal)
    open = priority queue ordered by f = g + h
    g(start) = 0
    h(start) = misplaced (start, goal)
    while (open not empty)
        take state with smallest f
        if state == ~~path~~ goal
            return path
        for each neighbor state
            g = g(parent) + 1
            h = manhattan(neighbor, goal)
            f = g + h
            Add neighbor to open
        return no solution

function manhattan(state, goal):
    dist = 0
    for each tile in state:
        find position in goal
        add row_diff + col_diff to distance
    return distance.
```

Output :
Solution found in 2 moves

(1, 2, 3)
(4, 5, 6)
(0, 7, 8)

(1, 2, 3)
(4, 5, 6)
(7, 0, 8)

(1, 2, 3)
(4, 5, 6)
(7, 8, 0)