U23AI021

Lab assignment 07
Artificial Intelligence

Harshil Andhariya
u23ai021@coed.svnit.ac.in

Q1)

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

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

bool depthLimitedSearchMaze(vector<vector<int>>& maze, int
x, int y, int depth, int limit, vector<pair<int, int>>&
path) {
    if (depth > limit) return false;
    if (x < 0 || y < 0 || x >= maze.size() || y >=
maze[0].size() || maze[x][y] == 1) return false;
    if (maze[x][y] == 9) {
        path.push_back({x, y});
        return true;
    }

    maze[x][y] = 1;
    path.push_back({x, y});

    if (depthLimitedSearchMaze(maze, x + 1, y, depth + 1,
limit, path)) return true;
    if (depthLimitedSearchMaze(maze, x - 1, y, depth + 1,
limit, path)) return true;
    if (depthLimitedSearchMaze(maze, x, y + 1, depth + 1,
limit, path)) return true;
    if (depthLimitedSearchMaze(maze, x, y - 1, depth + 1,
limit, path)) return true;

    path.pop_back();
    return false;
}
```

```
bool depthLimitedSearchPuzzle(vector<vector<int>>& puzzle,
int depth, int limit, int x, int y,
vector<vector<vector<int>>>& path) {
    if (depth > limit) return false;
    if (puzzle == goal) {
        path.push_back(puzzle);
        return true;
    }

    path.push_back(puzzle);

    int dx[] = { -1, 1, 0, 0 };
    int dy[] = { 0, 0, -1, 1 };

    for (int i = 0; i < 4; i++) {
        int nx = x + dx[i];
        int ny = y + dy[i];

        if (nx >= 0 && nx < 3 && ny >= 0 && ny < 3) {
            swap(puzzle[x][y], puzzle[nx][ny]);
            if (depthLimitedSearchPuzzle(puzzle, depth + 1,
limit, nx, ny, path)) return true;
            swap(puzzle[x][y], puzzle[nx][ny]);
        }
    }

    path.pop_back();
    return false;
}

void printMazePath(const vector<pair<int, int>>& path) {
    cout << "Maze Path: ";
    for (const auto& p : path) {
        cout << "(" << p.first << ", " << p.second << ") ";
    }
    cout << endl;
}
```

```
void printPuzzlePath(const vector<vector<vector<int>>>&
path) {
    cout << "8-Puzzle Path:" << endl;
    for (const auto& state : path) {
        for (const auto& row : state) {
            for (int val : row) {
                cout << val << " ";
            }
            cout << endl;
        }
        cout << "-----" << endl;
    }
}

int main() {
    vector<vector<int>> maze = {
        {0, 1, 0, 0},
        {0, 0, 0, 1},
        {1, 0, 1, 0},
        {0, 0, 9, 1}
    };
    int limit = 10;
    vector<pair<int, int>> mazePath;
    if (depthLimitedSearchMaze(maze, 0, 0, 0, limit,
mazePath)) {
        cout << "Maze DLS: Found" << endl;
        printMazePath(mazePath);
    } else {
        cout << "Maze DLS: Not Found" << endl;
    }

    vector<vector<int>> puzzle = {
        {1, 2, 3},
        {4, 5, 6},
        {0, 7, 8}
    };
    limit = 15;
    vector<vector<vector<int>>> puzzlePath;
```

```
    if (depthLimitedSearchPuzzle(puzzle, 0, limit, 2, 0,
puzzlePath)) {
        cout << "8-Puzzle DLS: Found" << endl;
        printPuzzlePath(puzzlePath);
    } else {
        cout << "8-Puzzle DLS: Not Found" << endl;
    }

    return 0;
}
```

Maze DLS: Found

Maze Path: (0, 0) (1, 0) (1, 1) (2, 1) (3, 1) (3, 2)

8-Puzzle DLS: Found

8-Puzzle Path:

1 2 3

4 5 6

0 7 8

---

1 2 3

0 5 6

4 7 8

---

0 2 3

1 5 6

4 7 8

---

1 2 3

0 5 6

4 7 8

---

0 2 3

1 5 6

4 7 8

---

1 2 3

0 5 6

4 7 8

---

0 2 3

1 5 6

4 7 8

---

1 2 3

0 5 6

4 7 8

---

0 2 3

1 5 6

4 7 8

---

1 2 3

0 5 6

4 7 8

---

0 2 3

1 5 6

4 7 8

---

1 2 3

0 5 6

4 7 8

---

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

Q2)

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

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

bool depthLimitedSearchMaze(vector<vector<int>>& maze, int
x, int y, int depth, int limit, vector<pair<int, int>>&
path) {
    if (depth > limit) return false;
    if (x < 0 || y < 0 || x >= maze.size() || y >=
maze[0].size() || maze[x][y] == 1) return false;
    if (maze[x][y] == 9) {
```

```
        path.push_back({x, y});
        return true;
    }

    maze[x][y] = 1;
    path.push_back({x, y});

    if (depthLimitedSearchMaze(maze, x + 1, y, depth + 1,
limit, path)) return true;
    if (depthLimitedSearchMaze(maze, x - 1, y, depth + 1,
limit, path)) return true;
    if (depthLimitedSearchMaze(maze, x, y + 1, depth + 1,
limit, path)) return true;
    if (depthLimitedSearchMaze(maze, x, y - 1, depth + 1,
limit, path)) return true;

    path.pop_back();
    return false;
}

bool depthLimitedSearchPuzzle(vector<vector<int>>& puzzle,
int depth, int limit, int x, int y,
vector<vector<vector<int>>>& path) {
    if (depth > limit) return false;
    if (puzzle == goal) {
        path.push_back(puzzle);
        return true;
    }

    path.push_back(puzzle);

    int dx[] = { -1, 1, 0, 0 };
    int dy[] = { 0, 0, -1, 1 };

    for (int i = 0; i < 4; i++) {
        int nx = x + dx[i];
        int ny = y + dy[i];
```

```
        if (nx >= 0 && nx < 3 && ny >= 0 && ny < 3) {
            swap(puzzle[x][y], puzzle[nx][ny]);
            if (depthLimitedSearchPuzzle(puzzle, depth + 1,
limit, nx, ny, path)) return true;
            swap(puzzle[x][y], puzzle[nx][ny]);
        }
    }

    path.pop_back();
    return false;
}

bool iterativeDeepeningSearchMaze(vector<vector<int>>& maze,
int startX, int startY, vector<pair<int, int>>& path) {
    int limit = 0;
    while (true) {
        vector<vector<int>> mazeCopy = maze;
        path.clear();
        if (depthLimitedSearchMaze(mazeCopy, startX, startY,
0, limit, path)) return true;
        limit++;
    }
}

bool iterativeDeepeningSearchPuzzle(vector<vector<int>>&
puzzle, int startX, int startY, vector<vector<vector<int>>>&
path) {
    int limit = 0;
    while (true) {
        vector<vector<int>> puzzleCopy = puzzle;
        path.clear();
        if (depthLimitedSearchPuzzle(puzzleCopy, 0, limit,
startX, startY, path)) return true;
        limit++;
    }
}

void printMazePath(const vector<pair<int, int>>& path) {
```

```
    cout << "Maze Path: ";
    for (const auto& p : path) {
        cout << "(" << p.first << ", " << p.second << ") ";
    }
    cout << endl;
}

void printPuzzlePath(const vector<vector<vector<int>>>&
path) {
    cout << "8-Puzzle Path:" << endl;
    for (const auto& state : path) {
        for (const auto& row : state) {
            for (int val : row) {
                cout << val << " ";
            }
            cout << endl;
        }
        cout << "-----" << endl;
    }
}

int main() {
    vector<vector<int>> maze = {
        {0, 1, 0, 0},
        {0, 0, 0, 1},
        {1, 0, 1, 0},
        {0, 0, 9, 1}
    };
    vector<pair<int, int>> mazePath;
    if (iterativeDeepeningSearchMaze(maze, 0, 0, mazePath))
{
        cout << "Maze IDS: Found" << endl;
        printMazePath(mazePath);
    } else {
        cout << "Maze IDS: Not Found" << endl;
    }

    vector<vector<int>> puzzle = {
```

```
        {1, 2, 3},
        {4, 5, 6},
        {0, 7, 8}
    };
    vector<vector<vector<int>>> puzzlePath;
    if (iterativeDeepeningSearchPuzzle(puzzle, 2, 0,
puzzlePath)) {
        cout << "8-Puzzle IDS: Found" << endl;
        printPuzzlePath(puzzlePath);
    } else {
        cout << "8-Puzzle IDS: Not Found" << endl;
    }

    return 0;
}
```

Output Clear

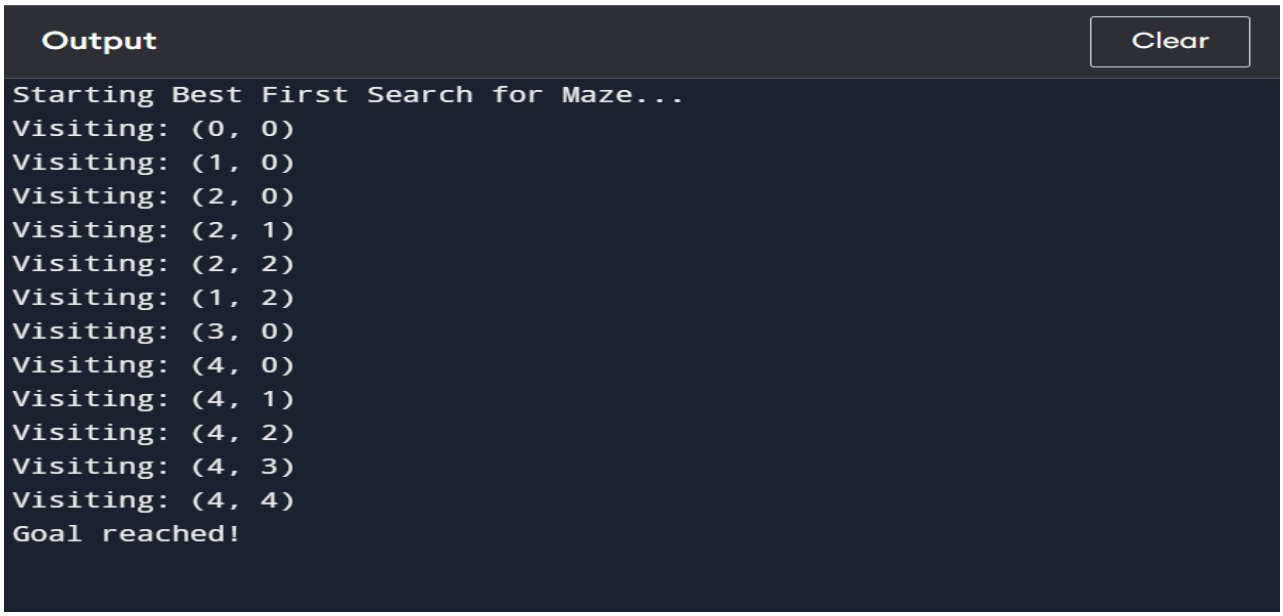
```
Maze IDS: Found
Maze Path: (0, 0) (1, 0) (1, 1) (2, 1) (3, 1) (3, 2)
8-Puzzle IDS: Found
8-Puzzle Path:
1 2 3
4 5 6
0 7 8
-----
1 2 3
4 5 6
7 0 8
-----
1 2 3
4 5 6
7 8 0
-----
```

3)

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

vector<vector<int>> maze = {
    {0, 1, 0, 0, 0},
    {0, 1, 0, 1, 0},
    {0, 0, 0, 1, 0},
    {0, 1, 1, 1, 0},
    {0, 0, 0, 0, 0}
};

pair<int, int> goal = {4, 4};

int heuristic(pair<int, int> a, pair<int, int> b) {
    return abs(a.first - b.first) + abs(a.second -
b.second);
}

void bestFirstSearchMaze(pair<int, int> start) {
    priority_queue<pair<int, pair<int, int>>,
vector<pair<int, pair<int, int>>>, greater<>> pq;
    vector<vector<bool>> visited(maze.size(),
vector<bool>(maze[0].size(), false));

    pq.push({heuristic(start, goal), start});
    visited[start.first][start.second] = true;

    while (!pq.empty()) {
        auto current = pq.top().second;
        pq.pop();

        cout << "Visiting: (" << current.first << ", " <<
current.second << ")" << endl;

        if (current == goal) {
            cout << "Goal reached!" << endl;
            return;
```

```
        }

        // Explore neighbors
        vector<pair<int, int>> directions = {{-1, 0}, {1,
0}, {0, -1}, {0, 1}};
        for (auto dir : directions) {
            int x = current.first + dir.first;
            int y = current.second + dir.second;

            if (x >= 0 && x < maze.size() && y >= 0 && y <
maze[0].size() && maze[x][y] == 0 && !visited[x][y]) {
                visited[x][y] = true;
                pq.push({heuristic({x, y}, goal), {x, y}});
            }
        }
    }

    cout << "Goal not reachable!" << endl;
}

int main() {
    pair<int, int> start = {0, 0};
    cout << "Starting Best First Search for Maze..." <<
endl;
    bestFirstSearchMaze(start);
    return 0;
}
```

Ouput:



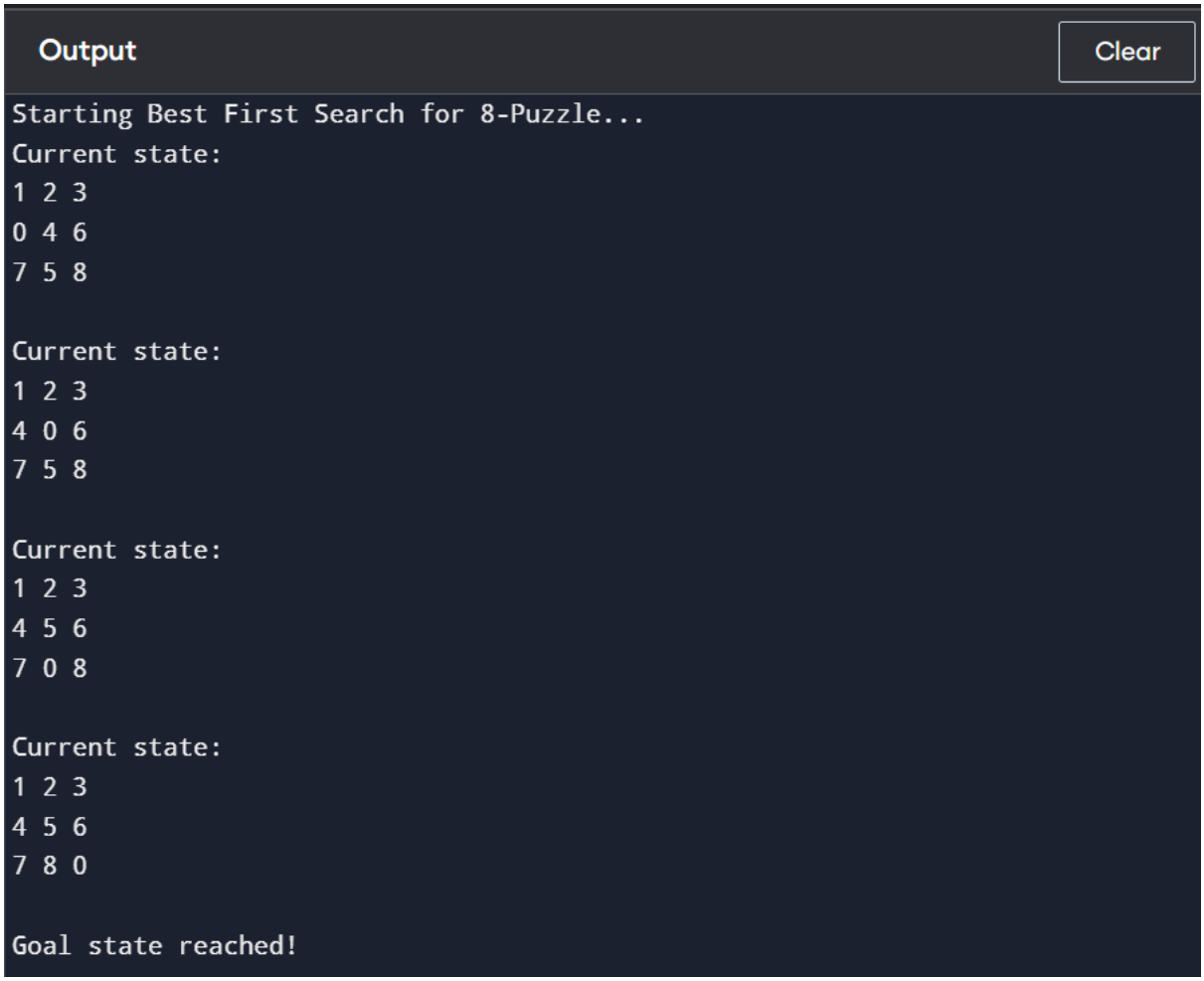
3b)

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>
#include <cmath>
#include <algorithm>

using namespace std;

int heuristic(vector<int>& state, vector<int>& goal) {
    int distance = 0;
    for (int i = 0; i < 9; i++) {
        if (state[i] != 0) {
            int goalPos = find(goal.begin(), goal.end(),
state[i]) - goal.begin();
            distance += abs(i / 3 - goalPos / 3) + abs(i % 3
- goalPos % 3);
        }
    }
```

```
    return distance;
}

void bestFirstSearch8Puzzle(vector<int>& start, vector<int>&
goal) {
    priority_queue<pair<int, vector<int>>, vector<pair<int,
vector<int>>>, greater<>> pq;
    unordered_set<string> visited;

    pq.push({heuristic(start, goal), start});
    visited.insert(string(start.begin(), start.end()));

    while (!pq.empty()) {
        auto current = pq.top().second;
        pq.pop();

        cout << "Current state: "<<endl;
        for (int i = 0; i < 9; i++) {
            cout << current[i] << " ";
            if ((i + 1) % 3 == 0) cout << endl;
        }
        cout << endl;

        if (current == goal) {
            cout << "Goal state reached!" << endl;
            return;
        }

        int zeroPos = find(current.begin(), current.end(),
0) - current.begin();
        int x = zeroPos / 3, y = zeroPos % 3;

        vector<pair<int, int>> directions = {{-1, 0}, {1,
0}, {0, -1}, {0, 1}};
        for (auto dir : directions) {
            int nx = x + dir.first, ny = y + dir.second;
            if (nx >= 0 && nx < 3 && ny >= 0 && ny < 3) {
                vector<int> nextState = current;
```

```
                swap(nextState[zeroPos], nextState[nx * 3 +
ny]);

                string nextStateStr =
string(nextState.begin(), nextState.end());
                if (visited.find(nextStateStr) ==
visited.end()) {
                    visited.insert(nextStateStr);
                    pq.push({heuristic(nextState, goal),
nextState});
                }
            }
        }
    }

    cout << "Goal state not reachable!" << endl;
}

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

    cout << "Starting Best First Search for 8-Puzzle..." <<
endl;
    bestFirstSearch8Puzzle(start, goal);
    return 0;
}
```

Output:


Output
Clear
Starting Best First Search for 8-Puzzle...
Current state:
1 2 3
0 4 6
7 5 8
Current state:
1 2 3
4 0 6
7 5 8
Current state:
1 2 3
4 5 6
7 0 8
Current state:
1 2 3
4 5 6
7 8 0
Goal state reached!