Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology Pune-37

(An autonomous Institute of Savitribai Phule Pune University)

Department of Computer Engineering

Division	C
Batch	2
GR-no	12111180
Rollno	17
Name	Divya Mahajan

Title: Implementation of Informed strategies.

Description:

Implementation of 8-Puzzle problem using Best-First Search:

Best-First Search is a search algorithm that explores the most promising nodes based on a heuristic evaluation function. It only considers the estimated cost from the current state to the goal state, and not the level.

Here, we are solving the 8-Puzzle problem, considering the number of misplaced tiles as the heuristic value.

Following are the steps:

1. A state is represented as a 2D matrix.

2. Create a priority queue (min-heap) to store states along with their heuristic values.

3. Enqueue the initial state along with its heuristic value into the priority queue.

4. While the priority queue is not empty:

    - Dequeue the state with the lowest heuristic value from the priority queue.
    - If the state is the goal state, you have found the solution.
    - Else, generate all possible next states by sliding the empty tile in valid directions.
    - Calculate the Misplaced tiles heuristic for the state.
    - Enqueue the state along with its heuristic value into the priority queue.

Implementation of 8-Puzzle problem using A*:

A* is an informed search algorithm that combines both the actual cost to reach a node (known as the "g-cost") and the estimated cost from the node to the goal (known as the "h-cost").

It uses an evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost to reach

the current node and h(n) is the estimated cost from the current node to the goal.

Following are the steps:

1. A state is represented as a 2D matrix.

2. Create a priority queue (min-heap) to store states along with their level and heuristic values.

3. Enqueue the initial state along with its level and heuristic value into the priority queue.

4. While the priority queue is not empty:

    - Dequeue the state with the lowest level + heuristic value from the priority queue.
    - If the state is the goal state, you have found the solution.
    - Else, generate all possible next states by sliding the empty tile in valid directions.
    - Calculate the Misplaced tiles heuristic for the state.
    - Enqueue the state along with its level and heuristic value into the priority queue.

Code:

- Best First Search

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

int countMisplaced(vector<vector<int>> &curr, vector<vector<int>>
&final)
{
    int count = 0;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
```

```
            if (curr[i][j] != 0 && curr[i][j] != final[i][j])
                count++;
        }
    }
    return count;
}

pair<int, int> findBlank(vector<vector<int>> vec)
{
    pair<int, int> ans;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (vec[i][j] == 0)
                ans = {i, j};
        }
    }
    return ans;
}

void printState(vector<vector<int>> state)
{
    for (auto row : state)
    {
        for (auto el : row)
        {
            cout << el << " ";
        }
        cout << endl;
    }
    cout << endl;
}

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

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

    priority_queue<pair<int, vector<vector<int>>>, vector<pair<int,
vector<vector<int>>>>, greater<>> pq;
    int h = countMisplaced(start, final);
    pq.push({h, start});

    //              R  D   L  U
    int delrow[] = {0, 1, 0, -1};
    int delcol[] = {1, 0, -1, 0};

    while (!pq.empty())
    {
        vector<vector<int>> curr = pq.top().second;
        pq.pop();
        printState(curr);

        h = countMisplaced(curr, final);
        cout << "h = " << h << endl
             << endl;

        if (h == 0)
        {
            cout << "Solution found!" << endl;
            break;
        }

        pair<int, int> blank = findBlank(curr);
        int row = blank.first;
        int col = blank.second;

        for (int i = 0; i < 4; i++)
        {
            int nrow = row + delrow[i];
            int ncol = col + delcol[i];
            if (nrow >= 0 && nrow < 3 && ncol >= 0 && ncol < 3)
            {
```

```
                swap(curr[row][col], curr[nrow][ncol]);
                h = countMisplaced(curr, final);
                pq.push({h, curr});
                swap(curr[row][col], curr[nrow][ncol]);
            }
        }
    }
    return 0;
}
```

- A*

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

int countMisplaced(vector<vector<int>> &curr, vector<vector<int>>
&final)
{
    int count = 0;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (curr[i][j] != 0 && curr[i][j] != final[i][j])
                count++;
        }
    }
    return count;
}

pair<int, int> findBlank(vector<vector<int>> vec)
{
    pair<int, int> ans;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (vec[i][j] == 0)
                ans = {i, j};
        }
    }
    return ans;
}

void printState(vector<vector<int>> state)
{
    for (auto row : state)
    {
        for (auto el : row)
```

```
        {
            cout << el << " ";
        }
        cout << endl;
    }
    cout << endl;
}

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

    priority_queue<pair<int, vector<vector<int>>>, vector<pair<int,
vector<vector<int>>>>, greater<>> pq;
    int h = countMisplaced(start, final);
    int g = 0;
    pq.push({g + h, start});

    //              R  D   L  U
    int delrow[] = {0, 1, 0, -1};
    int delcol[] = {1, 0, -1, 0};

    while (!pq.empty())
    {
        vector<vector<int>> curr = pq.top().second;
        pq.pop();
        printState(curr);

        h = countMisplaced(curr, final);
        cout << "g = " << g << endl;
        cout << "h = " << h << endl
             << endl;

        if (h == 0)
```

```
        {
            cout << "Solution found!" << endl;
            break;
        }

        g++;

        pair<int, int> blank = findBlank(curr);
        int row = blank.first;
        int col = blank.second;

        for (int i = 0; i < 4; i++)
        {
            int nrow = row + delrow[i];
            int ncol = col + delcol[i];
            if (nrow >= 0 && nrow < 3 && ncol >= 0 && ncol < 3)
            {
                swap(curr[row][col], curr[nrow][ncol]);
                h = countMisplaced(curr, final);
                pq.push({g + h, curr});
                swap(curr[row][col], curr[nrow][ncol]);
            }
        }
    }
    return 0;
}
```

Screenshots/Output:

Best First Search:

```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign3\"
2 8 3
1 6 4
7 0 5

h = 4

2 8 3
1 0 4
7 6 5

h = 3

2 0 3
1 8 4
7 6 5

h = 3

0 2 3
1 8 4
7 6 5

h = 2

1 2 3
0 8 4
7 6 5

h = 1

1 2 3
8 0 4
7 6 5

h = 0

Solution found!
PS D:\AI\Lab\Assign3>
```

A*:

```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign3\"
2 8 3
1 6 4
7 0 5

g = 0
h = 4

2 8 3
1 0 4
7 6 5

g = 1
h = 3

2 0 3
1 8 4
7 6 5

g = 2
h = 3

0 2 3
1 8 4
7 6 5

g = 3
h = 2

1 2 3
0 8 4
7 6 5

g = 4
h = 1

1 2 3
8 0 4
7 6 5

g = 5
h = 0

Solution found!
PS D:\AI\Lab\Assign3>
```