Assignment Name: 01 and 02 Assignment name : 8 puzzle problem and Best-First search in Graph representation problem

CSE-0408 Summer 2021

Mushiur Rahman Any
UG02-47-18-019
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email : mushiurany@gmail.com

***Abstract*—Code 1 : In practice, an incomplete heuristic search nearly always finds better solutions if it is allowed to search deeper, The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state. Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile. The empty space cannot move diagonally and can take only one step at a time. Index-python**

Code 2 :Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored. Index Terms—C++,Python

n

***Index Terms*—**

I. Introduction

Code 1 : The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

Code 2 :Breadth First Search (BFS) is an algorithm for traversing or searching layerwise in tree or graph data structures.If we consider searching as a form of traversal in a graph, an uninformed search algorithm would blindly traverse to the next node in a given manner without considering the cost associated with that step. An informed search, like Best first search, on the other hand would use an evaluation function to decide which among the various available nodes is the most promising (or 'BEST') before traversing to that node. The Best first search uses the concept of a Priority queue and heuristic search. To search the graph space, the BFS method uses two lists for tracking the traversa

II. Literature Review

Code 1 : Sadikov and Bratko (2006) studied the suitability of pessimistic and optimistic heuristic functions for a real-time search in the 8-puzzle. They discovered that pessimistic functions are more suitable. They also observed the pathology, which was stronger with the pessimistic heuristic function. However, they did not study the influence of other factors on the pathology or provide any analysis of the gain of a deeper search.

Code 2: Best First Search is a merger of Breadth First Search . Best First Search is implemented using the priority queue. while Breadth First Search arrives at a solution without search guaranteed that the procedure does not get caught. Best First Search, being a mixer of these two, licenses exchanging between paths. At each stage the nodes among the created ones, the best appropriate node is chosen for facilitating expansion, might be this node have a place to a similar level or different, hence can flip between Depth First and Breadth First Search [3]. It is also known as greedy search. Time complexity is O(bd) and space complexity is O(bd), where b is branching factor and d is solution depth

III. Proposed Methodology

Code 1: The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

Code 2 :Begin the search algorithm, by knowing the key which is to be searched. Once the key/element to be searched is decided the searching begins with the root (source) first.

Code 1 :

```python
DIRECTIONS = {"D": [-1, 0], "U": [1, 0], "R": [0, -1], "L": [0, 1]}
END = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

# unicode
left_down_angle = '\u2514'
right_down_angle = '\u2518'
right_up_angle = '\u2510'
left_up_angle = '\u250C'

middle_junction = '\u253C'
top_junction = '\u252C'
bottom_junction = '\u2534'
right_junction = '\u2524'
left_junction = '\u251C'

bar = Style.BRIGHT + Fore.CYAN + '\u2502' + Fore.RESET + Style.RESET_ALL
dash = '\u2500'

first_line = Style.BRIGHT + Fore.CYAN + left_up_angle + dash + dash + dash + top
middle_line = Style.BRIGHT + Fore.CYAN + left_junction + dash + dash + dash + mi
last_line = Style.BRIGHT + Fore.CYAN + left_down_angle + dash + dash + dash + bc


def print_puzzle(array):
    print(first_line)
    for a in range(len(array)):
        for i in array[a]:
            if i == 0:
                print(bar, Back.RED + ' ' + Back.RESET, end=' ')
            else:
                print(bar, i, end=' ')
        print(bar)
[redacted]
```

a) fig :1

```python
        self.current_node = current_node
        self.previous_node = previous_node
        self.g = g
        self.h = h
        self.dir = dir

    def f(self):
        return self.g + self.h


def get_pos(current_state, element):
    for row in range(len(current_state)):
        if element in current_state[row]:
            return (row, current_state[row].index(element))


def euclidianCost(current_state):
    cost = 0
    for row in range(len(current_state)):
        for col in range(len(current_state[0])):
            pos = get_pos(END, current_state[row][col])
            cost += abs(row - pos[0]) + abs(col - pos[1])
    return cost


def getAdjNode(node):
    listNode = []
```

b) fig :2

```python
        node = closedSet[str(node.previous_node)]
    branch.append({
        'dir': '',
        'node': node.current_node
    })
    branch.reverse()

    return branch


def main(puzzle):
    open_set = {str(puzzle): Node(puzzle, puzzle, 0, euclidianCost(puzzle), "")}
    closed_set = {}

    while True:
        test_node = getBestNode(open_set)
        closed_set[str(test_node.current_node)] = test_node

        if test_node.current_node == END:
            return buildPath(closed_set)

        adj_node = getAdjNode(test_node)
        for node in adj_node:
            if str(node.current_node) in closed_set.keys() or str(node.current_node) in oper
                str(node.current_node)].f() < node.f():
                continue
            open_set[str(node.current_node)] = node

        del open_set[str(test_node.current_node)]
[redacted]
```

c) fig :3


```python
        node = closedSet[str(node.previous_node)]
    branch.append({
        'dir': '',
        'node': node.current_node
    })
    branch.reverse()

    return branch


:f main(puzzle):
    open_set = {str(puzzle): Node(puzzle, puzzle, 0, euclidianCost(
    closed_set = {}

    while True:
        test_node = getBestNode(open_set)
        closed_set[str(test_node.current_node)] = test_node

        if test_node.current_node == END:
            return buildPath(closed_set)

        adj_node = getAdjNode(test_node)
        for node in adj_node:
            if str(node.current_node) in closed_set.keys() or str(n
                str(node.current_node)].f() < node.f():
                continue
            open_set[str(node.current_node)] = node

        del open_set[str(test_node.current_node)]
```


d) fig :4

```
        del open_set[str(test_node.current_node)]


if __name__ == '__main__':
    br = main([[1, 2, 3],
               [8, 6, 0],
               [7, 5, 4]])

    print('total steps : ', len(br) - 1)
    print()
    print(dash + dash + right_junction, "INPUT", left_junction
    for b in br:
        if b['dir'] != '':
            letter = ''
            if b['dir'] == 'U':
                letter = 'UP'
            elif b['dir'] == 'R':
                letter = "RIGHT"
            elif b['dir'] == 'L':
                letter = 'LEFT'
            elif b['dir'] == 'D':
                letter = 'DOWN'
            print(dash + dash + right_junction, letter, left_j
        print_puzzle(b['node'])
        print()

    print(dash + dash + right_junction, 'ABOVE IS THE OUTPUT',
```

e) fig :5

: Code 2 :

```
#include <bits/stdc++.h>
using namespace std;
int M,N;
int parent[100];
int cost [1000][10000];
int find(int i)
{

        while (parent[i] != i)
        i = parent[i];
    return i;
}


void union1(int i, int j)
{
        int s = find(i);
        int r = find(j);
        parent[s] = r;
}


void BFsMST()
{
        int mincost = 0;
        int edge_count = 0;
        while (edge_count < M - 1)
    {
                int min = INT_MAX, s = -1, r = -1;
                for (int i = 0; i < M; i++) {
                        for (int j = 0; j < M; j++)
            {
                                if (find(i) != find(j) && cost[i][j] < min)
                                {
                                        min = cost[i][j];
                                        s = i;
```

f) fig :1

```
                                        r = j;
                                }
                        }
                }

                union1(s, r);
        cout<<"Edge "<<edge_count++<<":("<<s<<" "<<r<<") cost:"<<min<<e
                mincost += min;
        }
        cout<<endl<<"Minimum cost= "<<mincost;
}


int main()
{



    //freopen("input.txt","h",stdin);

    //cin>>M>>;

        M=8;
        N=9;
    for(int i=0;i<M;i++)
    {
        for(int j=0; j<M; j++)
        {
            cost[i][j]= INT_MAX;
        }
    }

    /*for(int j = 0;j<N;j++)
    {
        int p,q;
        cin>>p>>q;
```

g) fig :2

```
        N=9;
    for(int i=0;i<M;i++)
    {
        for(int j=0; j<M; j++)
        {
            cost[i][j]= INT_MAX;
        }
    }

    /*for(int j = 0;j<N;j++)
    {
        int p,q;
        cin>>p>>q;
        cin>>cost[p][q];
        //cost[q][p];
    }*/
  cost[0][1]=1;
  cost[0][2]=2;
  cost[2][3]=3;
  cost[3][4]=4;
  cost[3][5]=5;
  cost[5][4]=100;
  cost[5][6]=7;
  cost[6][7]=101;
  cost[4][7]=8;
    for (int i = 0; i < M; i++)
        parent[i] = i;

        /// Print the solution
        BFsMST();
        return 0;
  }
```

h) fig :3

IV. Conclusion and Future Work

Code 1 : We tested our code to see how many states it would take to get from the current state to the goal state, and we came up with seven.

Code 2 :The BFS algorithm is useful for analyzing thenodes in a graph and constructing the shortest path of traversing through these.

ACKNOWLEDGMENT

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

REFERENCES

[1] Code 1 :
] Piltaver, R., Lustrek, M., and Gams, M. (2012). The pathology of heuristic ˘ search in the 8-puzzle. Journal of Experimental and Theoretical Artificial Intelligence, 24(1), 65-94

[2] Code 2 : Daniel Carlos Guimarães Pedronette received a B.Sc. in computer science (2005) from the State University of São Paulo (Brazil) and the M.Sc. degree in computer science (2008) from the University of Campinas (Brazil). He got his doctorate in computer science at the same university in 2012.