Assignment No 1: 8 Ball Puzzle

CSE-0408 Summer 2021

Name : Bm Pasha
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email : bmpasha72@ gmail.com

***Abstract*——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.**

As discussed earlier, Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. Breadth-First Search in tree and graph is almost the same. The only difference is that the graph may contain cycles, so we may traverse to the same node again

n

***Index Terms*—Language : Python.**

I. Introduction

The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state. 8-Puzzle is an interesting game which requires a player to move blocks one at a time to solve a picture or a particular pattern. In this article I will be showing you how to write an intelligent program that could solve 8-Puzzle automatically using the A* algorithm using Python and PyGame. Instead of a picture, we will use a pattern of numbers as shown in the figure, that is the final state. If you need to go through the A* algorithm theory or 8-Puzzle

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree (See method 2 of this post). The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex. For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.

II. Literature Review

The study of heuristics in human decision-making was developed in the 19708 and the 19805 by the psychologists Amos Tvcrsky and Daniel Kahneman although the concept had been originally introduced by the Nobel laureate Herbext A. Simon. whose original. primary object of research was problem solving that i showed.

The two variants of Best First Search are Greedy Best First Search and A* Best First Search. Greedy BFS: Algorithm selects the path which appears to be the best, it can be known as the combination of depth-first search and breadth first search. Greedy BFS makes use of Heuristic function and search and allows us to take advantages of both algorithms. A* BFS: Is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.).

III. Bfs Algorithm

Step 1: Choose the starting node and insert it into queue Step 2: Find the vertices that have direct edges with the vertex(node) Step 3: Insert all the vertices found in step 3 into queue Step 4: Remove the first vertex(node) in queue Step 5: Continue this process until all the vertices are visited.

IV. Algorithm for 8 Ball Puzzle

Solving 8-Puzzle manually varies from person to person. To solve it by computer or AI, we need a bit of a basic understanding of how it works to get the Goal node. Following are the steps: 1.Get the current state of the scenario (refers to the board or game in real world). 2.Find the available moves and their cost. 3.Choose the move with the least cost and set it as the current state. 4.Check if it matches the goal state, if yes terminate, if no move to step 1. In the code, our agent (program) will look for an empty space ('0') in a state and then which moves are allowed and have the least cost. As a result it will move towards the goal which is our final state.

V. Which Techniques

The 8-puzzle is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing. The more general n-puzzle is a classical problem which can be solved using graph search techniques.

VI. Rules of solving puzzle

Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile. The empty space can only move in four directions (Movement of empty space) 1. Up 2. Down 3. Right or 4. Left The empty space cannot move diagonally and can take only one step at a time.

VII. Conclusion

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

i am testing my code to seeing that how many states it would take to get from the current state to the goal state, i am trying many of moves and that's works . Puzzles are also good for the brain. Studies have shown that doing jigsaw puzzles can improve cognition and visual-spatial reasoning. The act of putting the pieces of a puzzle together requires concentration and improves short-term memory and problem solving.

Acknowledgment

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

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

VIII. BFS Code

In [ ]:

```
import networkx as nx
import matplotlib.pyplot as plt
from collections import deque
import random

def CreateGraph(node, edge):
    G = nx.Graph()
    for i in range(1, node+1):
        G.add_node(i)
    for i in range(edge):
        u, v = random.randint(1, node
        G.add_edge(u, v)
    return G

def DrawGraph(G, color):
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels = Tru

def DrawIteratedGraph(G,col_val):
    pos = nx.spring_layout(G)
    color = ["green", "blue", "yellow
    values = []
    for node in G.nodes():
        values.append(color[col_val[n
    nx.draw(G, pos, with_labels = Tru

def BFS(start):
    queue = deque()
    queue.append(start)
    visited[start] = True
    level[start] = 0

    while queue:
        u = queue.popleft()
        print(u, " -> ", end = "")
        for v in G.adj[u]:
            if not visited[v]:
                queue.append(v)
                visited[v] = True
                level[v] = level[u] +

        DrawIteratedGraph(G, level)
        plt.title('From {}:'.format(u
        plt.title('Level {}:'.format(
        plt.show()

    print("End")

if __name__ == "__main__":

    print("Enter no of Node")
    node = int(input())
    print("Enter no of Edges")
    edge = int(input())

    G = CreateGraph(node, edge)
```

IX. 8 Ball Puzzle

In [3]:

```
from copy import deepcopy
from colorama import Fore, Back, S

DIRECTIONS = {"D": [-1, 0], "U": [
END = [[1, 2, 3], [8, 0, 4], [7, 6

# 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 +
dash = '\u2500'

first_line = Style.BRIGHT + Fore.C
sh + dash + right_up_angle + Fore.
middle_line = Style.BRIGHT + Fore.
sh + dash + dash + right_junction
last_line = Style.BRIGHT + Fore.CY
sh + dash + dash + right_down_angl


def print_puzzle(array):
    print(first_line)
    for a in range(len(array)):
        for i in array[a]:
            if i == 0:
                print(bar, Back.RE
            else:
                print(bar, i, end=
        print(bar)
        if a == 2:
            print(last_line)
        else:
            print(middle_line)


class Node:
    def __init__(self, current_nod
        self.current_node = curren
        self.previous_node = previ
        self.g = g
        self.h = h
        self.dir = dir

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


def buildPath(closedSet):
    node = closedSet[str(END)]
    branch = list()

    while node.dir:
        branch.append({
            'dir': node.dir,
            'node': node.current_node
        })
        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, puzzl
    closed_set = {}

    while True:
        test_node = getBestNode(open_set)
        closed_set[str(test_node.current_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
                str(node.current_node)].f() < n
                continue
            open_set[str(node.current_node)] =

        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"
    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_junction + dash + dash)
    print_puzzle(b['node'])
    print()

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

```
total steps :  3
```

—┤ INPUT ├—

1	2	3
8	6	
7	5	4

—┤ UP ├—

1	2	3
8	6	4
7	5	

—┤ RIGHT ├—

1	2	3
8	6	4
7		5

—┤ DOWN ├—

1	2	3
8		4
7	6	5

—┤ ABOVE IS THE OUTPUT ├—