Practical - 5

October 26, 2023

1 Write a program to implement 8-puzzle problem by using A* algorithm.

[1]:
```
from typing import Optional, List
import random, copy, time, heapq
import numpy as np

class State:
    def __init__(self, N = 8) -> None:
        self.pos = [1,2,3,5,6,0,7,8,4]
        self.heuristic: int  = 0
        self.g: int = 0
        self.total_cost: int = self.heuristic + self.g
    def calculate_heuristic(self,goal_state):
        heuristic=0
        for num in range(1,9):
            distance=abs(self.pos.index(num) - goal_state.index(num))
            i=int(distance/3)
            j=int(distance%3)
            heuristic=heuristic+i+j
        self.heuristic = heuristic
    def update_total_cost(self) -> None:
        self.g += 1
        self.total_cost = self.heuristic + self.g
    def __lt__(self, other):
        return self.total_cost < other.total_cost
    def __eq__(self, other):
        return self.pos == other
    def __str__(self):
        return str(self.pos[0:3])+'\n'+str(self.pos[3:6])+'\n'+str(self.pos[6:9])

class AStarAlgorithm:
    goal_state = [1,2,3,5,8,6,0,7,4]
    @staticmethod
    def generate_random_state(N: int):
       state = State(N=N)
       state.calculate_heuristic(AStarAlgorithm.goal_state)
```

```
        return state
    @staticmethod
    def generate_child_states(state: State, states: List[State], visited_states:␣
↪List[State]) -> List[State]:
        children = []
        x = state.pos.index(0)
        i = int(x / 3)
        j = int(x % 3)
        legal_actions= AStarAlgorithm.find_legal_actions(i,j)
        for action in legal_actions:
            new_state = copy.deepcopy(state)
            if action == 'U':
                new_state.pos[x], new_state.pos[x-3] = new_state.pos[x-3],␣
↪new_state.pos[x]
            elif action == 'D':
                new_state.pos[x], new_state.pos[x+3] = new_state.pos[x+3],␣
↪new_state.pos[x]
            elif action == 'L':
                new_state.pos[x], new_state.pos[x-1] = new_state.pos[x-1],␣
↪new_state.pos[x]
            elif action == 'R':
                new_state.pos[x], new_state.pos[x+1] = new_state.pos[x+1],␣
↪new_state.pos[x]
            new_state.calculate_heuristic(AStarAlgorithm.goal_state)
            new_state.update_total_cost()
            if new_state not in states and new_state not in visited_states:
                children.append(new_state)
        return children
    @staticmethod
    def find_legal_actions(i,j):
        legal_action = ['U', 'D', 'L', 'R']
        if i == 0:  # up is disable
            legal_action.remove('U')
        elif i == 2:  # down is disable
            legal_action.remove('D')
        if j == 0:
            legal_action.remove('L')
        elif j == 2:
            legal_action.remove('R')
        return legal_action
    @staticmethod
    def is_goal_reached(state: State):
        if state.pos == AStarAlgorithm.goal_state:
            return True
        return False
class RunAStarAlgorithm:
    @staticmethod
```

```
    def run_a_star(N: Optional[int] = 8):
        states: List[State] = []
        visited_states: List[State] = []
        goal_reached: bool = False
        steps: int = 1
        start_time: float = time.time()
        initial_state: State = AStarAlgorithm.generate_random_state(N=N)
        states.append(initial_state)
        heapq.heapify(states)
        while len(states) != 0 and not goal_reached:
            curr_state: State = heapq.heappop(states)
            print(f'Current state: f(n) = {curr_state.total_cost}, g(n) =␣
↪{curr_state.g}, h(n) = {curr_state.heuristic}\n{curr_state}')
            visited_states.append(curr_state)
            if AStarAlgorithm.is_goal_reached(curr_state):
                print()
                print(f'GOAL STATE = \n{curr_state}')
                goal_reached = True
                break
            else:
                child_states: List[State] = AStarAlgorithm.
↪generate_child_states(curr_state, states, visited_states)
                states.extend(child_states)
                heapq.heapify(states)
            steps += 1
        else:
            print('SOLUTION NOT FOUND!')
        finish_time: float = time.time()
        time_taken: float = finish_time - start_time
        if not goal_reached:
            steps -= 1
        print('############################################')
        print(f'Total Steps = {steps}')
        print(f'Solution =\n{curr_state}')
        print(f'Time taken = {time_taken} seconds')
        return [steps, time_taken, goal_reached]
a_star_output = RunAStarAlgorithm.run_a_star(N=8)
```

```
Current state: f(n) = 0, g(n) = 0, h(n) = 3
[1, 2, 3]
[5, 6, 0]
[7, 8, 4]
Current state: f(n) = 3, g(n) = 1, h(n) = 2
[1, 2, 3]
[5, 0, 6]
[7, 8, 4]
Current state: f(n) = 3, g(n) = 2, h(n) = 1
```

```
[1, 2, 3]
[5, 8, 6]
[7, 0, 4]
Current state: f(n) = 3, g(n) = 3, h(n) = 0
[1, 2, 3]
[5, 8, 6]
[0, 7, 4]

GOAL STATE =
[1, 2, 3]
[5, 8, 6]
[0, 7, 4]
##########################################
Total Steps = 4
Solution =
[1, 2, 3]
[5, 8, 6]
[0, 7, 4]
Time taken = 0.0002219676971435547 seconds
```