MMAI5000A_Assignment1

September 29, 2020

1 MMAI 5000A AI Fundamentals – Assignment 1

Search Algorithms and intro to Python

1.1 Submission

The assignment is due on October 14 at 8:30. Submit standard Python file (i.e. `.py`) containing the required code.

1.2 Tasks

- Implement Depth-First Search (DFS)
  - It is very similar to BFS, but uses a Last-In-First-Out (LIFO) stack instead of a First-In-First-Out (FIFO) queue.
  - Hint: Check the list section in the Python datastructures docs.
- Run `GreedySearch()` with Kitchener as root node (`initi_state`) and Listowel as `goal_name`.
  - Build a new search tree, i.e. don't use the one generated by `map2searchtree.py`.
  - The nodes in this search tree needs to include `heuristic`, i.e. the heuristic function. E.g. add `node['heuristic']` (similar to `node['weight']`) to `node`.
  - Only include the locations listed below. The heuristic function is given after each node name and corresponds to the red lines on slide 51 ("Greedy Search: Map example") of lecture 2:
    - Kitchener : 130
    - Guelph : 160
    - Drayton : 100
    - New Hamburg : 110
    - Stratford : 100
    - St. Marys : 130
    - Mitchell : 100
- Record and return the path (i.e. sequence of nodes) the search algorithm took to reach the goal.
  - Hint: Add `node['path'] = []` to `node` and then record the parent and parent's parent (and so on) when a node is added to the `frontier`. `list.extend()` might prove useful. See the Python datastructures docs for more info about Python lists.

1.3 Code provided

1.3.1 Search tree

`map2searchtree.py` – get the latest version from Canvas

1.3.2 Search algorithms

Breadth-First Search (BFS)

[ ]:

```
# Breadth-first search
def BFS(init_state, goal_name):
    """Breadth-First Search (BFS)

    Arguments
    ---------
    init_state : the root node of a search tree
    goal_name  : A string, the name of a node, e.g. tree.childrend[0].name
    """

    frontier = [init_state]
    explored = []

    while len(frontier):
        state = frontier.pop()  # dequeue
        explored.append(state['name'])

        if state['name'] == goal_name:
            return True

        for child in state['children']:
            if child['name'] not in explored:
                # enqueue: insert node at the beginning
                frontier.insert(0, child)

    return False
```

Uniform Cost Search

[2]:

```
# UCS helper
def find_min_weight(frontier):
    # Helper func to find min weight/distance
    min_weight_i = 0
    if len(frontier) > 1:
        min_weight = frontier[min_weight_i]['weight']
        for i, state in enumerate(frontier):
            if state['weight'] < min_weight:
                min_weight_i = i
                min_weight = state['weight']
    return min_weight_i

def UCS(init_state, goal_name):
    """Uniform Cost Search (UCS)

    Arguments
    ---------
    init_state : the root node of a search tree
    goal_name  : A string, the name of a node, e.g. tree.childrend[0].name
    """
    frontier = [init_state]
    explored = []

    while len(frontier):
        # next state -> state w lowest cost/weight/distance
        state = frontier.pop(find_min_weight(frontier))
        explored.append(state['name'])

        if state['name'] == goal_name:
            return True

        for child in state['children']:
            if child['name'] not in explored:
                frontier.append(child)

    return False
```

Greedy Search

[ ]:

```
# Greedy helper
def find_min_heuristic(frontier):
    # Helper func to find min of h (the heuristic function)
    min_h_i = 0
    if len(frontier) > 1:
        min_h = frontier[min_h_i]['heuristic']
        for i, state in enumerate(frontier):
            if state['heuristic'] < min_h:
                min_h_i = i
                min_h = state['heuristic']
    return min_h_i


def GreedySearch(init_state, goal_name):
    frontier = [init_state]
    explored = []

    while len(frontier):
        state = frontier.pop(find_min_heuristic(frontier))
        explored.append(state['name'])

        if state['name'] == goal_name:
            return True
        for child in state['children']:
            if child['name'] not in explored:
                frontier.append(child)

    return False
```

Good luck!