CMPUT366: Assignment 1

(Fall 2021)

Material Coverage: Lecture 1 to 4

Algorithms Involved: Dijkstra and A*

Assignment Weight: 12%

Mark Distribution: 50% for correct code implementation; 50% for answering questions about your results

Goal of this assignment:

Your goal is to implement Dijkstra's algorithm and A* for solving pathfinding problems

- We will consider a grid environment where each action in the four cardinal directions (north, south, east, and west) has the cost of 1.0 and each action in one of the four diagonal directions has the cost of 1.5.
- Each search problem is defined by a video game map, a start location and a goal location.
- You will use a single map in our experiments; feel free to explore other maps if you want.

  Gridded_map = Map("dao-map/brc000d.map") in main.py

Python Environment Setup:

- Use of virtual environment is encouraged
  1. Download the assignment zip, unzip it
  2. Open a terminal at the unzipped folder
  3. virtualenv -p python3 venv (choose a name for your virtual environment)
  4. source venv/bin/activate
  5. pip install -r requirements.txt

Now the assignment environment is set up.

Implementing Dijkstra:

- S_o: start node
- S_g: goal node
- T(n): returns the children of node n
- g(n): cost from Sinit to n
- c(n, n’): cost from n to n’
- g(n) + c(n, n’): cost of n’
- OPEN(priority queue)
- CLOSED(python dictionary (hash_value: node))

```
Def Dijkstra($S_o$, $S_g$, T):
    OPEN.insert($S_o$, 0)
    CLOSED.add($S_o$)
    # node_expanded will be used for explaining the results later
    node_expanded = 0
    while OPEN is not empty:
        n = OPEN.pop()
        node_expanded +=1
        # goal found
        If n == $S_g$:
            Return Path

    for n’ in T(n):
        # n’ is not expanded before
        If n’ not in CLOSED:
            OPEN.insert(n’, g(n) + c(n, n’))
            CLOSED.Add(n’)
        # n’ is expanded before, but a cheaper path is found to n’
        If n’ in CLOSED & (g(n) + c(n, n’) < g(n’)):
            Update g(n’) to  g(n) + c(n, n’) in OPEN
            Update g(n’) to g(n) + c(n, n’) in CLOSED

    # no solution
    Return -1
```

FAQ:

1. This Assignment did not ask for a returning solution path. So you do not need to update parent of n’ if a cheaper path to n’ is found

2. Hash_value is a unique identification for a node. For example, you can use self.CLOSED[hash_value] to access a specific node in the CLOSED dictionary.

3. map.successors(n) can be used to access the children(or neighbors) of a node n.

4. You don’t need to apply heap on the CLOSED dictionary since you don’t need to pop out any value from.

In the OPEN list. Nodes can be sorted by their cost. (See the __lt__ method in State class, algorithms.py). You can easily use the python heap queue algorithm to turn the OPEN list to a priority queue.

Reference: https://docs.python.org/3/library/heapq.html

heapq (you may need to use it for OPEN):

This module provides an implementation of the heap queue algorithm, also known as the priority queue algorithm.

heapq.**heappush**(*heap*, *item*)

> Push the value *item* onto the *heap*, maintaining the heap invariant.

heapq.**heappop**(*heap*)

> Pop and return the smallest item from the *heap*, maintaining the heap invariant. If the heap is empty, IndexError is raised. To access the smallest item without popping it, use `heap[0]`.

heapq.**heapify**(*x*)

> Transform list *x* into a heap, in-place, in linear time.

Implementing A* Algorithm:

- S_o: start node
- S_g: goal node
- $h(n, S_g)$: heuristic function; estimates the cost to reach goal from node n
- current: current node (n)
- T(n): returns the children of node n
- g(n): cost from S_o to n
- c(n, n'): cost from n to n'
- g(n) + c(n, n'): cost of n'
- OPEN(priority queue)
- CLOSED(python dictionary)

```
Def A*($S_o$, $S_g$, h):

    OPEN.insert($S_o$, 0)
    while OPEN is not empty:
        n = OPEN.pop()
        # goal found
        If n == $S_g$:
            Return Path
        CLOSED.Add(n)
        # Loop through the node's children
        for n' in T(n):
            # If it is already in CLOSED, skip it
            if n' in CLOSED:
                continue
            if n' in OPEN:
                if g(n') > g(n) + c(n, n'):
                    G score of node n' = g(n) + c(n, n')
            else:
                # If it isn't in the open set, calculate the G and H score for n'
                G score of node n' = g(n) + c(n, n')
                H score of node n' = h(n', $S_g$)
                # Add it to OPEN
                OPEN.insert(n', G score of n' + H score of n')
    # no solution
    Return -1
```