Advanced Algorithms Lab-Min term

Prashanth.S
(19MID0020)

Code Structure

```
> def a_star_algorithm(starting_node, stop_node):⋯


> def get_neighbors(v):⋯


> def heuristic(n):⋯

> if __name__ == '__main__':⋯

```

Code Snippet

```
def a_star_algorithm(starting_node, stop_node):
    open_set = set(starting_node)
    closed_set = set()
    g = {}
    parents = {}
    g[starting_node] = 0
    parents[starting_node] = starting_node

    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n == stop_node or graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
```

```python
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)

        if n == None:
            print('The Path does not exist!')
            return None

        if n == stop_node:
            path = []

            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(starting_node)
            path.reverse()

            print('Final Path from vertex-0 to vertex-4 : {}'.format(path))
            return path

        open_set.remove(n)
        closed_set.add(n)

    print('Path does not exist!')
    return None
```

```python


def get_neighbors(v):
    if v in graph_nodes:
        return graph_nodes[v]
    else:
        return None


def heuristic(n):
    heurisitic_distances = {'0': 0, '1': 2, '2': 15, '3': 8,
                '4': 0, '5': 8, '6': 5, '7': 4, '8': 1, }
    return heurisitic_distances[n]

if __name__ == '__main__':
    graph_nodes = {
        '0': [('1', 4), ('7', 8)],
        '1': [('0', 4), ('7', 11), ('2', 8)],
        '2': [('1', 8), ('8', 2), ('5', 4), ('3', 7)],
        '3': [('2', 7), ('5', 14), ('4', 9)],
        '4': [('3', 9), ('5', 10)],
        '5': [('4', 10), ('3', 14), ('2', 4), ('6', 2)],
        '6': [('5', 2), ('8', 6), ('7', 1)],
        '7': [('0', 8), ('1', 11), ('8', 7), ('6', 1)],
        '8': [['2', 2], ['6', 6], ['7', 7]]
    }

    a_star_algorithm('0', '4')
```

Output

Final Path from vertex-0 to vertex-4 : ['0', '7', '6', '5', '4']

Code

```python
def a_star_algorithm(starting_node, stop_node):
    open_set = set(starting_node)
    closed_set = set()
    g = {}
    parents = {}
    g[starting_node] = 0
    parents[starting_node] = starting_node

    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n
):
                n = v

        if n == stop_node or graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)

        if n == None:
            print('The Path does not exist!')
            return None

        if n == stop_node:
            path = []

            while parents[n] != n:
```

```python
                path.append(n)
                n = parents[n]
            path.append(starting_node)
            path.reverse()

            print('Final Path from vertex-0 to vertex-
4 : {}'.format(path))
            return path

        open_set.remove(n)
        closed_set.add(n)

    print('Path does not exist!')
    return None


def get_neighbors(v):
    if v in graph_nodes:
        return graph_nodes[v]
    else:
        return None


def heuristic(n):
    heurisitic_distances = {'0': 0, '1': 2, '2': 15, '3': 8,
              '4': 0, '5': 8, '6': 5, '7': 4, '8': 1, }
    return heurisitic_distances[n]

if __name__ == '__main__':
    graph_nodes = {
        '0': [('1', 4), ('7', 8)],
        '1': [('0', 4), ('7', 11), ('2', 8)],
        '2': [('1', 8), ('8', 2), ('5', 4), ('3', 7)],
        '3': [('2', 7), ('5', 14), ('4', 9)],
        '4': [('3', 9), ('5', 10)],
        '5': [('4', 10), ('3', 14), ('2', 4), ('6', 2)],
        '6': [('5', 2), ('8', 6), ('7', 1)],
        '7': [('0', 8), ('1', 11), ('8', 7), ('6', 1)],
        '8': [['2', 2], ['6', 6], ['7', 7]]
    }

    a_star_algorithm('0', '4')
```