بسمه تعالی

گزارش تمرین پیاده سازی سری اول هوش مصنوعی

نام و نام خانوادگی: سینا مظاهری شماره دانشجویی: ۹۸۱۷۱۱۵۹

ما در ابتدا فرض نمودیم که محیطی که عامل ما یعنی ربات با آن سر و کار دارد یک محیط گسسته زمان و حالت هست و همچنین برای ربات کاملا شناخته شده و مشاهده پذیر می باشد. یعنی ربات از محل باتری و محل خود در نقشه اطلاع کامل دارد اما در مورد موانع تا زمانی که به مجاورت آن ها نرود آن ها را احساس نمی کند. در نتیجه جست جوی ما یک جست و جوی آگاهانه (A*) و یا ناآگاهانه (BFS در ادامه) خواهد بود.

تابع اکتشافی یا هیوریستیک ما در ۲ فایل برنامه ای که به شکل مجزا در تمرین آمده به دو شکل متفاوت است در یکی فاصله منهتن (Manhatan Distance) یا همان عبارت زیر

$$h(n) = |n.x - g.x| + |n.y - g.y|$$

به عنوان هیورستیک در نظر گرفته شده که به وضوح قابل قبول است زیرا این هزینه از راس تا هدف دقیقا متناظر با هزینه ای است که هدف و راس در یک زیر مستطیل از کل فضای حالت قرار گرفته باشند که در نتیجه آن $|n.x - g.x|$ حرکت افقی (فقط به راست یا فقط به چپ) و همچنین $|n.y - g.y|$ حرکت عمودی (فقط به بالا یا فقط به پایین) خواهد داشت و این دقیقا متناظر حالت بهینه است از طرفی در یک پیاده سازی دیگر فرض شده است که فاصله اقلیدسی (Euclidean Distance) یک تابع هیوریستیک است و در نتیجه چون این تابع طبق نامساوی مثلثی سازگار است در نتیجه قابل قبول نیز می باشد.

$$h(n) = \sqrt{(n.x - g.x)^2 + (n.y - g.y)^2}$$

نتایج اجرای الگوریتم با هیوریستیک منهتن بر روی فایل SampleRoom.xml به شکل زیر می باشد:

```
I've found an optimal path!!!

Step: 0

| | | |R| |*|
|*| |*| | | |
| | | |*| | |
| |*|*|B|*| |
| |*| | |*| |
| | | | | | |



Step: 1

| | | | | |*|
|*| |*|R| | |
| | | |*| | |
| |*|*|B|*| |
| |*| | |*| |
| | | | | | |



Step: 2

| | | | | |*|
|*| |*| |R| |
| | | |*| | |
| |*|*|B|*| |
| |*| | |*| |
| | | | | | |
```

File - informed_search

```
40 Step: 3
41
42 | | | | | |*|
43 |*| |*| | | |
44 | | | |*|R| |
45 | |*|*|B|*| |
46 | |*| | |*| |
47 | | | | | | |
48
49
50
51
52 Step: 4
53
54 | | | | | |*|
55 |*| |*| | | |
56 | | | |*| |R|
57 | |*|*|B|*| |
58 | |*| | |*| |
59 | | | | | | |
60
61
62
63
64 Step: 5
65
66 | | | | | |*|
67 |*| |*| | | |
68 | | | |*| | |
69 | |*|*|B|*|R|
70 | |*| | |*| |
71 | | | | | | |
72
73
74
75
76 Step: 6
77
78 | | | | | |*|
79 |*| |*| | | |
80 | | | |*| | |
```

```
 81 | |*|*|B|*| |
 82 | |*| | |*|R|
 83 | | | | | | |
 84
 85
 86
 87
 88 Step: 7
 89
 90 | | | | | |*|
 91 |*| |*| | | |
 92 | | | |*| | |
 93 | |*|*|B|*| |
 94 | |*| | |*| |
 95 | | | | | |R|
 96
 97
 98
 99
100 Step: 8
101
102 | | | | | |*|
103 |*| |*| | | |
104 | | | |*| | |
105 | |*|*|B|*| |
106 | |*| | |*| |
107 | | | | |R| |
108
109
110
111
112 Step: 9
113
114 | | | | | |*|
115 |*| |*| | | |
116 | | | |*| | |
117 | |*|*|B|*| |
118 | |*| | |*| |
119 | | | |R| | |
120
121
```

File - informed_search

```
122
123
124 Step: 10
125
126 | | | | | |*|
127 |*| |*| | | |
128 | | | |*| | |
129 | |*|*|B|*| |
130 | |*| |R|*| |
131 | | | | | | |
132
133
134
135
136 Step: 11
137
138 | | | | | |*|
139 |*| |*| | | |
140 | | | |*| | |
141 | |*|*|R|*| |
142 | |*| | |*| |
143 | | | | | | |
144
145
146
147
148
149 Process finished with exit code 0
150
```

۱۱ گام برای رسیدن به محل تغذیه پیدا شده که همان ۱۱ گام حالت بهینه می باشد. عکس کد پس از شرح کد آمده است. ما ابتدا پس از آماده سازی گره ها و توابع **expand** و **evaluation_funcion** و همچنین **action** و **result** طبق الگوریتم جست و جوی اول بهترین ابتدا یک صف اولویت برای **frontier** در نظر گرفتیم. سپس یک دیکشنری به نام **reached** که به ازای هر حالت گره دیده شده آن را در خود ذخیره می کند. ادامه الگوریتم مانند جست و جوی اول بهترین می باشد.

```python
import sys
import lxml.etree
import heapq
from math import fabs


class Node:
    def __init__(self, state, parent_node, path_cost, action):
        self.state = state
        self.parent_node = parent_node
        self.path_cost = path_cost
        self.action = action

    def get_state(self):
        return self.state

    def get_path_cost(self):
        return self.path_cost

    def get_parent_node(self):
        return self.parent_node


def expand(node, plan):
    state = node.get_state()
    all_children = set()
    for single_action in action(state, plan):
        the_tuple = result(node, single_action)
        cost = node.get_path_cost() + 1
        all_children.add(Node(the_tuple, node, cost, single_action))
    return all_children
```

```python
def result(node, action):
    state = node.get_state()
    the_tuple = [state[0], state[1]]
    if action == 'L':
        the_tuple[1] -= 1
    elif action == 'R':
        the_tuple[1] += 1
    elif action == 'U':
        the_tuple[0] -= 1
    else:
        the_tuple[0] += 1
    the_tuple.append("empty")
    return tuple(the_tuple)


def action(state, plan):
    all_actions = {'D', 'U', 'L', 'R'}
    state_y = state[0]
    state_x = state[1]
    length_y = len(plan)
    length_x = len(plan[0])
    if state_y + 1 == length_y or plan[state_y + 1][state_x].text == "obstacle":
        all_actions.difference_update({'D'})
    if state_y - 1 == -1 or plan[state_y - 1][state_x].text == "obstacle":
        all_actions.difference_update({'U'})
    if state_x + 1 == length_x or plan[state_y][state_x + 1].text == "obstacle":
        all_actions.difference_update({'R'})
    if state_x - 1 == -1 or plan[state_y][state_x - 1].text == "obstacle":
        all_actions.difference_update({'L'})
    return all_actions
```

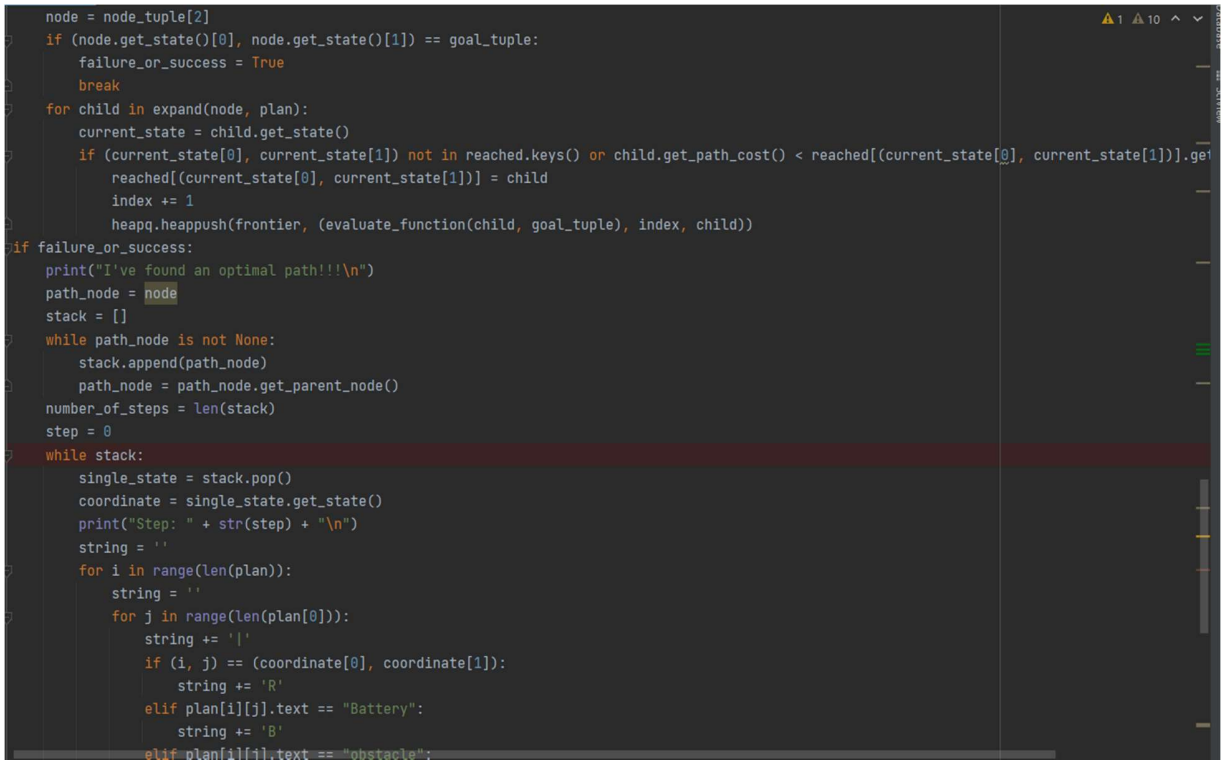
```
def evaluate_function(node, goal):
    node_state = node.get_state()
    return fabs(node_state[0] - goal[0]) + fabs(node_state[1] - goal[1]) + node.get_path_cost()




tree = lxml.etree.parse(sys.argv[1])
a_1 = tree.xpath('//cell[text()="robot"]')[0]
a_2 = a_1.getparent()
plan = a_2.getparent()
initial_state = (plan.index(a_2), a_2.index(a_1), "robot")
robot_node = Node(initial_state, None, 0, None)

a_1 = plan.xpath('//cell[text()="Battery"]')[0]
a_2 = a_1.getparent()
plan = a_2.getparent()
goal_tuple = (plan.index(a_2), a_2.index(a_1))
frontier = []
reached = {}
index = 0
heapq.heappush(frontier, (evaluate_function(robot_node, goal_tuple), index, robot_node))
reached[(initial_state[0], initial_state[1])] = robot_node
number_of_steps = 0
failure_or_success = False
while frontier:
    node_tuple = heapq.heappop(frontier)
    node = node_tuple[2]
    if (node.get_state()[0], node.get_state()[1]) == goal_tuple:
```

```
    node = node_tuple[2]
    if (node.get_state()[0], node.get_state()[1]) == goal_tuple:
        failure_or_success = True
        break
    for child in expand(node, plan):
        current_state = child.get_state()
        if (current_state[0], current_state[1]) not in reached.keys() or child.get_path_cost() < reached[(current_state[0], current_state[1])].get
            reached[(current_state[0], current_state[1])] = child
            index += 1
            heapq.heappush(frontier, (evaluate_function(child, goal_tuple), index, child))
if failure_or_success:
    print("I've found an optimal path!!!\n")
    path_node = node
    stack = []
    while path_node is not None:
        stack.append(path_node)
        path_node = path_node.get_parent_node()
    number_of_steps = len(stack)
    step = 0
    while stack:
        single_state = stack.pop()
        coordinate = single_state.get_state()
        print("Step: " + str(step) + "\n")
        string = ''
        for i in range(len(plan)):
            string = ''
            for j in range(len(plan[0])):
                string += '|'
                if (i, j) == (coordinate[0], coordinate[1]):
                    string += 'R'
                elif plan[i][j].text == "Battery":
                    string += 'B'
                elif plan[i][j].text == "obstacle":
```



```
    node = node_tuple[2]
    if (node.get_state()[0], node.get_state()[1]) == goal_tuple:
        failure_or_success = True
        break
    for child in expand(node, plan):
        current_state = child.get_state()
        if (current_state[0], current_state[1]) not in reached.keys() or child.get_path_cost() < reached[(current_state[0], current_state[1])].ge
            reached[(current_state[0], current_state[1])] = child
            index += 1
            heapq.heappush(frontier, (evaluate_function(child, goal_tuple), index, child))
if failure_or_success:
    print("I've found an optimal path!!!\n")
    path_node = node
    stack = []
    while path_node is not None:
        stack.append(path_node)
        path_node = path_node.get_parent_node()
    number_of_steps = len(stack)
    step = 0
    while stack:
        single_state = stack.pop()
        coordinate = single_state.get_state()
        print("Step: " + str(step) + "\n")
        string = ''
        for i in range(len(plan)):
            string = ''
            for j in range(len(plan[0])):
                string += '|'
                if (i, j) == (coordinate[0], coordinate[1]):
                    string += 'R'
                elif plan[i][j].text == "Battery":
                    string += 'B'
                elif plan[i][j].text == "obstacle":
```

فرض کنیم فایل SampleRoom.xml را تغییر دهیم یعنی محل ربات به خانه (1,5) برود و محل باتری به (5,0) برود (اندیس ها از صفر شروع می شوند) در اینصورت نتایج خروجی به شکل زیر می شود:

```
I've found an optimal path!!!

Step: 0

| | | | | |*|
|*| |*| | |R|
| | | |*| | |
| |*|*| |*| |
| |*| | |*| |
|B| | | | | |


Step: 1

| | | | | |*|
|*| |*| | | |
| | | |*| |R|
| |*|*| |*| |
| |*| | |*| |
|B| | | | | |


Step: 2

| | | | | |*|
|*| |*| | | |
| | | |*| | |
| |*|*| |*|R|
| |*| | |*| |
|B| | | | | |


```

```
40 Step: 3
41
42 | | | | | |*|
43 |*| |*| | | |
44 | | | |*| | |
45 | |*|*| |*| |
46 | |*| | |*|R|
47 |B| | | | | |
48
49
50
51
52 Step: 4
53
54 | | | | | |*|
55 |*| |*| | | |
56 | | | |*| | |
57 | |*|*| |*| |
58 | |*| | |*| |
59 |B| | | | |R|
60
61
62
63
64 Step: 5
65
66 | | | | | |*|
67 |*| |*| | | |
68 | | | |*| | |
69 | |*|*| |*| |
70 | |*| | |*| |
71 |B| | | |R| |
72
73
74
75
76 Step: 6
77
78 | | | | | |*|
79 |*| |*| | | |
80 | | | |*| | |
```

```
 81 | |*|*| |*| |
 82 | |*| | |*| |
 83 |B| | |R| | |
 84
 85
 86
 87
 88 Step: 7
 89
 90 | | | | | |*|
 91 |*| |*| | | |
 92 | | | |*| | |
 93 | |*|*| |*| |
 94 | |*| | |*| |
 95 |B| |R| | | |
 96
 97
 98
 99
100 Step: 8
101
102 | | | | | |*|
103 |*| |*| | | |
104 | | | |*| | |
105 | |*|*| |*| |
106 | |*| | |*| |
107 |B|R| | | | |
108
109
110
111
112 Step: 9
113
114 | | | | | |*|
115 |*| |*| | | |
116 | | | |*| | |
117 | |*|*| |*| |
118 | |*| | |*| |
119 |R| | | | | |
120
121
```

اگر فرض کنیم که هیوریستیک ما فاصله اقلیدسی باشد نتیجه آن چنان تفاوتی نمی کند. یا حتی اگر تابع هیوریستیک صفر باشد (در جست و جوی اول سطح پیاده شده) مسئله چون ابعاد کوچکی دارد بر روی منتیجه آن چنان تاثیر گذار نخواهد بود اما اگر ابعاد مسئله و فضای حالت رفته رفته بزرگتر شود کارایی الگوریتم های آگاهانه نسبت به ناآگاهانه پررنگ تر و پررنگ تر می شود. یک الگوریتم جست و جوی اول سطح نیز پیاده سازی شده که تنها تفاوت آن این است که تابع ارزیاب عمق درخت جست و جو را بر می گرداند به دیگر سخن تنها هزینه رفتن از حالت اولیه به آن حالت مورد نظر را داریم و هیچ برآوردی تا هدف نخواهیم داشت. همه ی این ها در ۳ فایل مختلف با اسامی مشخص به شکل کد مشخص شده و همچنین نتایجشان در درون فایل پی دی اف پیوست این گزارش آمده است.