AU 332 Artificial Intelligence: Principles and Techniques

By: WangChunhui (517021910047)

HW#: 1

September 23, 2019

I. INTRODUCTION

A. Purpose

The goal of this week's lab is to review the four methods of search including bfs,dfs,ucs and astar search.And it needs you to realize those algorithms with python.Besides that,you also need calculate the complexity or path with the medthods mentuone below.

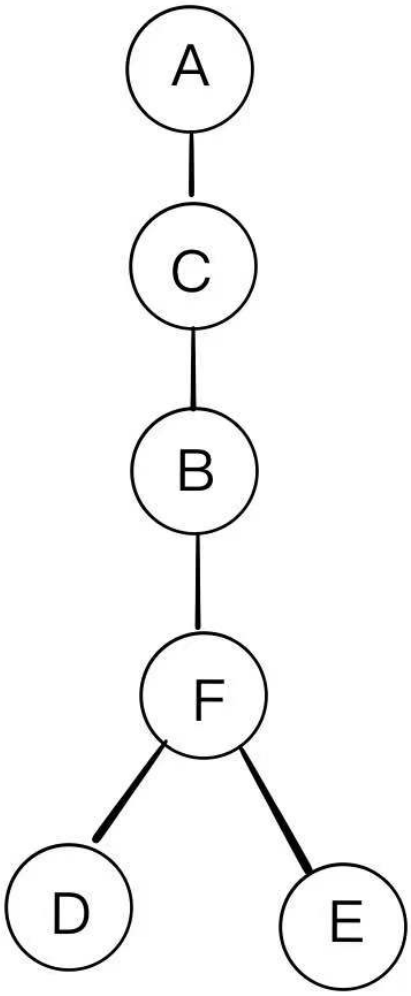
B. Equipment

There is a minimal amount of equipment to be used in this homework. The few requirements are listed below:

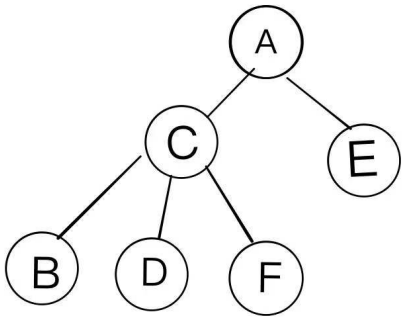
- python
- queue (priority queue and LIFO queue))

II. HOMEWORK

A. Graph Traversal



(a) DFS



(b) BFS

- bfs time complexy O(n*d)
- bfs space complexy O(n)
- dfs time complexy O(n*d)
- dfs space complexy O(n)

B. Uniform Cost Search Algorithm

- step1 'A': None
  [(0, 'A')]
- step2 'A': None, 'B': 'A'
  [(10, 'B')]
- step3 'A': None, 'B': 'A', 'D': 'A'
  [(10, 'B'), (20, 'D')]
- step4 'A': None, 'B': 'A', 'D': 'A', 'C': 'A'
  [(3, 'C'), (20, 'D'), (10, 'B')]
- step5'A': None, 'B': 'C', 'D': 'A', 'C': 'A'
  [(5, 'B'), (20, 'D'), (10, 'B')]
- step6 'A': None, 'B': 'C', 'D': 'A', 'C': 'A', 'E': 'C'
  [(5, 'B'), (18, 'E'), (10, 'B'), (20, 'D')]
- step7 'A': None, 'B': 'C', 'D': 'B', 'C': 'A', 'E': 'C'
  [(10, 'B'), (10, 'D'), (20, 'D'), (18, 'E')]

C. A* Algorithm

- heuistic funcition = Euclidean distances
- step1 [(1,2)]
  [(4,(2,2)),(5.12,(1,1)), (5.12,(1,3)),(6,(0,2))]

- step2 [(1,2),(2,2)]
  [(5.16,(2,1)),(5.16,(2,3)),(5.12,(1,1)), (5.12,(1,3)),(6,(0,2))]
- step3 [(1,2),(2,2),(1,1)]
  [(6.47,(1,0)),(7.09,(0,1)),(5.16,(2,1)),(5.16,(2,3)), (5.12,(1,3)),(6,(0,2))]
- step4 [(1,2),(2,2),(1,1),(1,3)]
  [(6.47,(1,4)),(7.09,(0,3)),(6.47,(1,0)),(7.09,(0,1)),(5.16,(2,1)),(5.16,(2,3)), (6,(0,2))]
- step5 [(1,2),(2,2),(1,1),(1,3),(2,1)]
  [(6.6,(2,0)),(6.47,(1,4)),(7.09,(0,3)),(6.47,(1,0)),(7.09,(0,1)),(5.16,(2,3)), (6,(0,2))]
- step6 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3)]
  [(6.6,(2,4)),(6.6,(2,0)),(6.47,(1,4)),(7.09,(0,3)),(6.47,(1,0)),(7.09,(0,1)), (6,(0,2))]
- step7 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2)]
  [(6.6,(2,4)),(6.6,(2,0)),(6.47,(1,4)),(7.09,(0,3)),(6.47,(1,0)),(7.09,(0,1))]
- step8 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4)]
  [(8.38,(0,4)),(6.6,(2,4)),(6.6,(2,0)),(7.09,(0,3)),(6.47,(1,0)),(7.09,(0,1))]
- step9 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0)]
  [(8.38,(0,0)),(8.38,(0,4)),(6.6,(2,4)),(6.6,(2,0)),(7.09,(0,3)),(7.09,(0,1))]
- step10 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0)]
  [(6.82,(3,0)),(8.38,(0,0)),(8.38,(0,4)),(6.6,(2,4)),(7.09,(0,3)),(7.09,(0,1))]
- step11 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4)]
  [(6.82,(3,4)),(6.82,(3,0)),(8.38,(0,0)),(8.38,(0,4)),(7.09,(0,3)),(7.09,(0,1))]
- step12 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0)]
  [(7.23,(4,0)),(6.82,(3,4)),(8.38,(0,0)),(8.38,(0,4)),(7.09,(0,3)),(7.09,(0,1))]
- step13 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4)]
  [(7.23,(4,4),(7.23,(4,0)),(8.38,(0,0)),(8.38,(0,4)),(7.09,(0,3)),(7.09,(0,1))]
- step14 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3)]
  [(7.23,(4,4),(7.23,(4,0)),(8.38,(0,0)),(8.38,(0,4)),(7.09,(0,1))]
- step15 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1)]
  [(7.23,(4,4),(7.23,(4,0)),(8.38,(0,0)),(8.38,(0,4))]
- step16 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1),(4,0)]
  [(7.41,(4,1)),(8,(5,0)),(7.23,(4,4),(8.38,(0,0)),(8.38,(0,4))]
- step17 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1),(4,0),(4,4)]
  [(7.41,(4,3)),(8,(5,4)),(7.41,(4,1)),(8,(5,0)),(8.38,(0,0)),(8.38,(0,4))]
- step18 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1),(4,0),(4,4),(4,3)]
  [(8,(4,2))(8,(5,3)),(8,(5,4)),(7.41,(4,1)),(8,(5,0)),(8.38,(0,0)),(8.38,(0,4))]
- step19 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1),(4,0),(4,4),(4,3),(4,1)]
  [(8,(5,1)),(8,(4,2)),(8,(5,3)),(8,(5,4)),(8,(5,0)),(8.38,(0,0)),(8.38,(0,4))]
- step20 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1),(4,0),(4,4),(4,3),(4,1),(5,1)]
  [(8,(5,2)),(8,(4,2)),(8,(5,3)),(8,(5,4)),(8,(5,0)),(8.38,(0,0)),(8.38,(0,4))]
- step20 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1),(4,0),(4,4),(4,3),(4,1),(5,1)]
  [(9.41,(6,1)),(8,(5,2)),(8,(4,2)),(8,(5,3)),(8,(5,4)),(8,(5,0)),(8.38,(0,0)),(8.38,(0,4))]
- step21 [(1,2),(2,2),(1,1),(1,3),(2,1),(2,3),(0,2),(1,4),(1,0),(2,0),(2,4),(3,0),(3,4),(0.3),(0,1),(4,0),(4,4),(4,3),(4,1),(5,1),(5,2)
  [(9.41,(6,1)),(8,(4,2)),(8,(5,3)),(8,(5,4)),(8,(5,0)),(8.38,(0,0)),(8.38,(0,4))]

D. Codes

```
while(1):
    if (goal == start):
        break
    else:
            path.append(goal)
            goal = came_from[goal]
path.append(start)

path.reverse()
```

(c) reconstructpath

```
came_from = {}
came_from[start] = None
### START CODE HERE ### (≈ 10 line of code)
sq=Queue()
sq.put(start)
searched = [start]
target=graph.edges[start]
parent=start

while goal not in target:
  for item in target:
     if item not in searched:
         came_from[item]=parent
         sq.put(item)
         searched.append(item)
  parent=sq.get()
  target=graph.edges[parent]
came_from[goal]=parent
### END CODE HERE ###
return came_from
```

(d) breadth first search

```
came_from = {}
came_from[start] = None
### START CODE HERE ### (≈ 10 line of code)
sq=LifoQueue()
sq.put(start)
searched=[]
searched.append(start)
target = graph.edges[start]
parent=start
while goal not in target:
    for item in target:
        if item not in searched:
            came_from[item] = parent
            sq.put(item)
            searched.append(item)
    parent=sq.get()
    target=graph.edges[parent]
    came_from[goal] = parent
```

(e) depth first search

```
Large graph
came from DFS {'S': None, 'A': 'S', 'C': 'S', 'E': 'K', 'L': 'C', 'I': 'L', 'J': 'L', 'K': 'J'}
path from DFS ['S', 'C', 'L', 'J', 'K', 'E']
came from BFS {'S': None, 'A': 'S', 'C': 'S', 'B': 'A', 'D': 'A', 'L': 'C', 'H': 'B', 'F': 'D', 'I': 'L', 'J': 'L', 'G': 'H', 'K': 'I', 'E': 'G'}
path from BFS ['S', 'A', 'B', 'H', 'G', 'E']
Small graph
came from DFS {'A': None, 'B': 'A', 'D': 'A', 'E': 'D'}
path from DFS ['A', 'D', 'E']
came from BFS {'A': None, 'B': 'A', 'D': 'A', 'C': 'B', 'E': 'D'}
path from BFS ['A', 'D', 'E']
```

(f) Results of bfs and dfs

```
### START CODE HERE ### (≈ 15 line of code)
sq=PriorityQueue()
visited=[]
sq._put((0,start))

while sq:
    parent=sq._get()[1]
    target = graph.edges[parent]
    targetcost=graph.edgeWeights[parent]
    if parent not in visited:
        visited.append(parent)
        if parent == goal:
         return came_from, cost_so_far
        for i in target:
            if (i not in cost_so_far) or (cost_so_far[i] > targetcost[target.index(i)] + cost_so_far[parent]):
                came_from[i]=parent
                cost_so_far[i]=(targetcost[target.index(i)]+cost_so_far[parent])
                sq._put((cost_so_far[i],i))
```

(g) uniform cost search

```
Small graph
came from UCS  {'A': None, 'B': 'A', 'D': 'A', 'C': 'B', 'E': 'D'}
cost form UCS  {'A': 0, 'B': 2, 'D': 4, 'C': 5, 'E': 7}
path from UCS  ['A', 'D', 'E']
Large graph
came from UCS  {'S': None, 'A': 'B', 'B': 'S', 'C': 'S', 'D': 'B', 'H': 'B', 'L': 'C', 'F': 'H', 'G': 'H', 'E': 'G', 'I': 'L', 'J': 'L'}
cost form UCS  {'S': 0, 'A': 5, 'B': 2, 'C': 3, 'D': 6, 'H': 3, 'L': 5, 'F': 6, 'G': 5, 'E': 7, 'I': 9, 'J': 9}
path from UCS  ['S', 'B', 'H', 'G', 'E']
```

(h) Results of ucs

```
    ### START CODE HERE ### (≈ 15 line of code)
    c1=graph.locations[current_node][0]-graph.locations[goal_node][0]
    c2=graph.locations[current_node][1]-graph.locations[goal_node][1]
    heuristic_value=(c1**2+c2**2)**(1/2)

    ### END CODE HERE ###
```

(i) heuristic

```
### START CODE HERE ### (≈ 15 line of code)
sq=PriorityQueue()
visited=[]
sq._put((0,start))

while sq:
    parent=sq._get()[1]
    target = graph.edges[parent]
    targetcost=graph.edgeWeights[parent]
    if parent not in visited:
        visited.append(parent)
        if parent == goal:
         return came_from, cost_so_far
        for i in target:
            if (i not in cost_so_far) or (cost_so_far[i] > targetcost[target.index(i)] + cost_so_far[parent]):
                came_from[i]=parent
                cost_so_far[i]=(targetcost[target.index(i)]+cost_so_far[parent])
                sq._put(((cost_so_far[i]+heuristic(graph, start, i)),i))



### END CODE HERE ###
```

(j) A star search

```
Small Graph
came from Astar  {'A': None, 'B': 'A', 'D': 'A', 'C': 'B', 'E': 'D'}
cost form Astar  {'A': 0, 'B': 2, 'D': 4, 'C': 5, 'E': 7}
path from Astar  ['A', 'D', 'E']
Large Graph
came from Astar  {'S': None, 'A': 'B', 'B': 'S', 'C': 'S', 'D': 'B', 'H': 'B', 'F': 'H', 'G': 'H', 'L': 'C', 'E': 'G', 'I': 'L', 'J': 'L', 'K': 'I'}
cost form Astar  {'S': 0, 'A': 5, 'B': 2, 'C': 3, 'D': 6, 'H': 3, 'F': 6, 'G': 5, 'L': 5, 'E': 7, 'I': 9, 'J': 9, 'K': 13}
path from Astar  ['S', 'B', 'H', 'G', 'E']
```

(k) Results of astar search

E. bonus solution

1. *bonus1 solution*

```
if start not in graph.edges:
    print(" the ", start, "not exist in",graph)
    return {}, {}
elif goal not in graph.edges:
    print(" the ", goal, "not exist in",graph)
    return {}, {}

else:
    sq=PriorityQueue()
    visited=[]
    sq._put((0,start))
```

(l) Extra credit 5.1

2. *bonus2 solution*

It can be proved that the graph satises the consistency of heuristics. because what I choose to be the heuristics is Euclidean distances.As we all know,if h(A)$\leqslant$ cost(A to C)+h(C). The heuristics is consist.In this graph,the cost of every step is 1,so cost(A to C) $\geqslant$ Euclidean distances(A to C).Because of Triangle law,h(A)-h(C) $\leqslant$ Euclidean distances(A to C)$\leqslant$ cost(A to C).