Artificial intelligence Lab

Subarna Khorshed

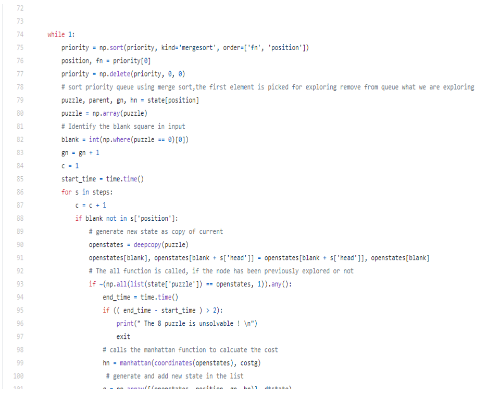
Ug02-48-18-010

1 Introduction

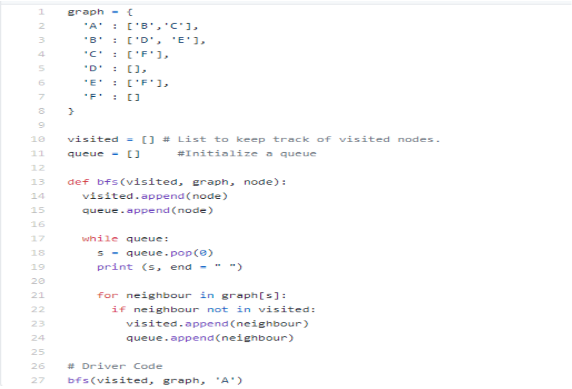
The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Our goal is to rearrange the blocks so that they are in order.

And in second assignment we will be solving problems for Bfs. Breadth-first search is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

2 Screenshot of code



```
while 1:
    priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])
    position, fn = priority[0]
    priority = np.delete(priority, 0, 0)
    # sort priority queue using merge sort,the first element is picked for exploring remove from queue what we are exploring
    puzzle, parent, gn, hn = state[position]
    puzzle = np.array(puzzle)
    # Identify the blank square in input
    blank = int(np.where(puzzle == 0)[0])
    gn = gn + 1
    c = 1
    start_time = time.time()
    for s in steps:
        c = c + 1
        if blank not in s['position']:
            # generate new state as copy of current
            openstates = deepcopy(puzzle)
            openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']], openstates[blank]
            # The all function is called, if the node has been previously explored or not
            if ~(np.all(list(state['puzzle']) == openstates, 1)).any():
                end_time = time.time()
                if (( end_time - start_time ) > 2):
                    print(" The 8 puzzle is unsolvable ! \n")
                    exit
                # calls the manhattan function to calcuate the cost
                hn = manhattan(coordinates(openstates), costg)
                 # generate and add new state in the list
                q = np.array([(openstates, position, gn, hn)], dtstate)
```



```python
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}

visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for neighbour in graph[s]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
bfs(visited, graph, 'A')
```

3 Acknowledgment

I would like to thank to our honourable teacher for helping me to understand this project.