

# Artificial Intelligence Assignments Codes

Sunday 28<sup>th</sup> May, 2023

## Listings

|   |   |    |
|---|---|----|
| 1 | Breadth first search . . . . .  | 2  |
| 2 | Depth first search . . . . .  | 3  |
| 3 | A* Algorithm in Python . . . . .  | 4  |
| 4 | Prims Algorithm in Python . . . . .   | 7  |
| 5 | N Queen Problem using backtracking global N in Python . . . . .                                     | 9  |
| 6 | N Queen Problem using Branch And Bound in Python . . . . .  | 12 |
| 7 | Implementation of Graph Colouring Using BackTracking in Python . . . . .                            | 16 |
| 8 | Develop an elementary catboat for any suitable customer interaction application in Python . . . . . | 18 |

```

1 from collections import deque
2
3 visited = deque([])
4 queue = deque([])
5
6 def breadth_first_search(visited, queue, graph, node):
7     visited.append(node)
8     queue.append(node)
9
10    while queue:          # Creating loop to visit each node
11        m = queue.popleft()
12        print(m, end = " ")
13
14        for neighbour in graph[m]:
15            if neighbour not in visited:
16                visited.append(neighbour)
17                queue.append(neighbour)
18
19 graph = {
20     '5': ['3', '7'],
21     '3': ['2', '4'],
22     '2': [],
23     '4': ['8'],
24     '7': ['6'],
25     '8': [],
26     '6': []
27 }
28
29 visited = deque([])
30 queue = deque([])
31
32 print("Following is the Breadth First Search")
33 breadth_first_search(visited, queue, graph, '5')
34 print(end="\n")

```

Listing 1: Breadth first search

```

1 graph = {
2     '5': ['3', '7'],
3     '3': ['2', '4'],
4     '2': [],
5     '4': ['8'],
6     '7': ['6'],
7     '8': [],
8     '6': []
9 }
10
11 visited = set()      # Set to keep track of visited nodes of graph.
12
13 def depth_first_search(visited, graph, node):
14     if node not in visited:
15         print(node, end = " ")
16         visited.add(node)
17
18         for neighbour in graph[node]:
19             depth_first_search(visited, graph, neighbour)
20
21 # Driver code for the program
22 print("Following is the Depth First Search")
23 depth_first_search(visited, graph, '5')

```

Listing 2: Depth first search

```

1 def aStarAlgo(start_node, stop_node):
2     open_set = set(start_node)
3     closed_set = set()
4
5     g = {}           # store distance from starting node
6     parents = {}     # parents contains an adjacency map of all nodes
7
8     # distance of starting node from itself is zero
9     g[start_node] = 0
10
11    # start_node is root node i.e it has no parent nodes
12    # so start_node is set to its own parent node
13    parents[start_node] = start_node
14
15    while len(open_set) > 0:
16        n = None
17        # node with lowest f() is found
18        for v in open_set:
19            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
20                n = v
21
22        if n == stop_node or Graph_nodes[n] == None:
23            pass
24
25        else:
26            for (m, weight) in get_neighbors(n):
27                # nodes 'm' not in first and last set are added to first
28                # n is set its parent
29
30                if m not in open_set and m not in closed_set:
31                    open_set.add(m)
32                    parents[m] = n
33                    g[m] = g[n] + weight
34
35                # for each node m, compare its distance from start i.e g(m) to the
36                # from start through n node
37
38            else:
39                if g[m] > g[n] + weight:
40                    # update g(m)

```

```

41         g[m] = g[n] + weight
42
43         # change parent of m to n
44         parents[m] = n
45
46         # if m in closed set, remove and add to open
47         if m in closed_set:
48             closed_set.remove(m)
49             open_set.add(m)
50
51     if n == None:
52         print('Path does not exist!')
53         return None
54
55     # if the current node is the stop_node
56     # then we begin reconstructin the path from it to the start_node
57
58     if n == stop_node:
59         path = []
60         while parents[n] != n:
61             path.append(n)
62             n = parents[n]
63
64         path.append(start_node)
65         path.reverse()
66         print('Path found: {}'.format(path))
67         return path
68
69     # remove n from the open_list, and add it to closed_list
70     # because all of his neighbors were inspected
71
72     open_set.remove(n)
73     closed_set.add(n)
74
75     print('Path does not exist!')
76     return None
77
78 # define fuction to return neighbor and its distance
79 # from the passed node
80 def get_neighbors(v):

```

```

81     if v in Graph_nodes:
82         return Graph_nodes[v]
83
84     else:
85         return None
86
87
88 #for simplicity we ll consider heuristic distances given
89 #and this function returns heuristic distance for all nodes
90 def heuristic(n):
91     H_dist = {
92         'A': 11,
93         'B': 6,
94         'C': 5,
95         'D': 7,
96         'E': 3,
97         'F': 6,
98         'G': 5,
99         'H': 3,
100        'I': 1,
101        'J': 0
102    }
103    return H_dist[n]
104
105 #Describe your graph here
106 Graph_nodes = {
107     'A': [('B', 6), ('F', 3)],
108     'B': [('A', 6), ('C', 3), ('D', 2)],
109     'C': [('B', 3), ('D', 1), ('E', 5)],
110     'D': [('B', 2), ('C', 1), ('E', 8)],
111     'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
112     'F': [('A', 3), ('G', 1), ('H', 7)],
113     'G': [('F', 1), ('I', 3)],
114     'H': [('F', 7), ('I', 2)],
115     'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
116 }
117
118 aStarAlgo('A', 'J')

```

Listing 3: A\* Algorithm in Python

```

1 import sys
2
3 class Graph():
4     def __init__(self, vertices):
5         self.V = vertices
6         self.graph = [[0 for column in range(vertices)]
7                        for row in range(vertices)]
8
9     def printMST(self, parent):
10        print("Edge \tWeight")
11        for i in range(1, self.V):
12            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])
13
14    def minKey(self, key, mstSet):
15
16        # Initialize min value
17        min = sys.maxsize
18
19        for v in range(self.V):
20            if key[v] < min and mstSet[v] == False:
21                min = key[v]
22                min_index = v
23
24        return min_index
25
26    def primMST(self):
27
28        key = [sys.maxsize] * self.V
29        parent = [None] * self.V
30        key[0] = 0
31        mstSet = [False] * self.V
32
33        parent[0] = -1
34
35        for cout in range(self.V):
36            u = self.minKey(key, mstSet)
37
38            mstSet[u] = True
39            for v in range(self.V):
40                if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:

```

```

41         key[v] = self.graph[u][v]
42         parent[v] = u
43
44         self.printMST(parent)
45
46 g = Graph(5)
47
48 g.graph = [
49     [0, 2, 0, 6, 0],
50     [2, 0, 3, 8, 5],
51     [0, 3, 0, 0, 7],
52     [6, 8, 0, 0, 9],
53     [0, 5, 7, 9, 0]
54 ]
55
56 g.primMST();

```

Listing 4: Prims Algorithm in Python



```

1 # Python3 program to solve N Queen
2 # Problem using backtracking global N
3
4 N = 4
5
6 def printSolution(board):
7     for i in range(N):
8         for j in range(N):
9             print(board[i][j], end = " ")
10        print()
11
12
13 # A utility function to check if a queen can
14 # be placed on board[row][col]. Note that this
15 # function is called when "col" queens are
16 # already placed in columns from 0 to col -1.
17 # So we need to check only left side for
18 # attacking queens
19
20 def isSafe(board, row, col):
21     # Check this row on left side
22     for i in range(col):
23         if board[row][i] == 1:
24             return False
25
26     # Check upper diagonal on left side
27     for i, j in zip(range(row, -1, -1),
28                     range(col, -1, -1)):
29         if board[i][j] == 1:
30             return False
31
32     # Check lower diagonal on left side
33     for i, j in zip(range(row, N, 1),
34                     range(col, -1, -1)):
35         if board[i][j] == 1:
36             return False
37     return True
38
39 def solveNQUtil(board, col):
40

```

```

41     # base case: If all queens are placed
42     # then return true
43     if col >= N:
44         return True
45
46     # Consider this column and try placing
47     # this queen in all rows one by one
48
49     for i in range(N):
50         if isSafe(board, i, col):
51             # Place this queen in board[i][col]
52             board[i][col] = 1
53
54             # recur to place rest of the queens
55             if solveNQUtil(board, col + 1) == True:
56                 return True
57
58             # If placing queen in board[i][col]
59             # doesn't lead to a solution, then
60             # queen from board[i][col]
61             board[i][col] = 0
62
63     # if the queen can not be placed in any row in
64     # this column col then return false
65
66     return False
67
68 # This function solves the N Queen problem using
69 # Backtracking. It mainly uses solveNQUtil() to
70 # solve the problem. It returns false if queens
71 # cannot be placed, otherwise return true and
72 # placement of queens in the form of 1s.
73 # note that there may be more than one
74 # solutions, this function prints one of the
75 # feasible solutions.
76
77 def solveNQ():
78     board = [
79         [0, 0, 0, 0],
80         [0, 0, 0, 0],

```

```

81     [0, 0, 0, 0],
82     [0, 0, 0, 0]
83 ]
84
85 if solveNQUtil(board, 0) == False:
86     print ("Solution does not exist")
87     return False
88
89 printSolution(board)
90 return True
91
92 # Driver Code
93 solveNQ()

```

Listing 5: N Queen Problem using backtracking global N in Python

```

1  '''
2  Python3 program to solve N Queen Problem
3  using Branch or Bound
4  '''
5
6  N = 8
7
8  # A utility function to print solution
9  def printSolution(board):
10     for i in range(N):
11         for j in range(N):
12             print(board[i][j], end = " ")
13         print()
14
15     '''
16     A Optimized function to check if
17     a queen can be placed on board[row][col]
18     '''
19     def isSafe(row, col, slashCode, backslashCode,
20               rowLookup, slashCodeLookup,
21               backslashCodeLookup):
22         if (slashCodeLookup[slashCode[row][col]] or
23             backslashCodeLookup[backslashCode[row][col]] or
24             rowLookup[row]):
25             return False
26         return True
27
28     '''
29     A recursive utility function
30     to solve N Queen problem
31     '''
32
33     def solveNQueensUtil(board, col, slashCode, backslashCode,
34                           rowLookup, slashCodeLookup,
35                           backslashCodeLookup):
36         '''
37         base case: If all queens are
38         placed then return True
39         '''
40         if(col >= N):

```

```

41     return True
42 for i in range(N):
43     if(isSafe(i, col, slashCode, backslashCode,
44             rowLookup, slashCodeLookup,
45             backslashCodeLookup)):
46         '''
47         Place this queen in board[i][col]
48         '''
49         board[i][col] = 1
50         rowLookup[i] = True
51         slashCodeLookup[slashCode[i][col]] = True
52         backslashCodeLookup[backslashCode[i][col]] = True
53
54         '''
55         recur to place rest of the queens
56         '''
57         if(solveNQueensUtil(board, col + 1,
58                             slashCode, backslashCode,
59                             rowLookup, slashCodeLookup,
60                             backslashCodeLookup)):
61             return True
62
63         '''
64         If placing queen in board[i][col]
65         doesn't lead to a solution, then backtrack
66         '''
67
68         # Remove queen from board[i][col]
69         board[i][col] = 0
70         rowLookup[i] = False
71         slashCodeLookup[slashCode[i][col]] = False
72         backslashCodeLookup[backslashCode[i][col]] = False
73
74     '''
75     If queen can not be place in any row in
76     this column col then return False
77     '''
78     return False
79
80 '''

```

```

81 This function solves the N Queen problem using
82 Branch or Bound. It mainly uses solveNQueensUtil()to
83 solve the problem. It returns False if queens
84 cannot be placed,otherwise return True or
85 prints placement of queens in the form of 1s.
86 Please note that there may be more than one
87 solutions,this function prints one of the
88 feasible solutions.
89 '''
90
91 def solveNQueens():
92     board = [
93         [0 for i in range(N)]
94         for j in range(N)
95     ]
96
97     # helper matrices
98     slashCode = [
99         [0 for i in range(N)]
100        for j in range(N)
101    ]
102
103    backslashCode = [
104        [0 for i in range(N)]
105        for j in range(N)
106    ]
107
108    # arrays to tell us which rows are occupied
109    rowLookup = [False] * N
110
111    # keep two arrays to tell us
112    # which diagonals are occupied
113    x = 2 * N - 1
114
115    slashCodeLookup = [False] * x
116    backslashCodeLookup = [False] * x
117
118    # initialize helper matrices
119    for rr in range(N):
120        for cc in range(N):

```

```

121         slashCode[rr][cc] = rr + cc
122         backslashCode[rr][cc] = rr - cc + 7
123
124     if(solveNQueensUtil(board, 0, slashCode, backslashCode,
125                         rowLookup, slashCodeLookup,
126                         backslashCodeLookup) == False):
127         print("Solution does not exist")
128         return False
129
130     # solution found
131     printSolution(board)
132     return True
133
134 # Driver Code
135 solveNQueens()

```

Listing 6: N Queen Problem using Branch And Bound in Python

```

1 # Python3 program for the above approach
2
3 # Number of vertices in the graph
4 # define 4 X 4
5
6 # check if the colored
7 # graph is safe or not
8
9 def isSafe(graph, color):
10     # check for every edge
11     for i in range(4):
12         for j in range(i + 1, 4):
13             if (graph[i][j] and color[j] == color[i]):
14                 return False
15     return True
16
17 # This function solves the m Coloring
18 # problem using recursion. It returns
19 # false if the m colours cannot be assigned,
20 # otherwise, return true and prints
21 # assignments of colours to all vertices.
22 # Please note that there may be more than
23 # one solutions, this function prints one
24 # of the feasible solutions.
25
26 def graphColoring(graph, m, i, color):
27
28     # if current index reached end
29     if (i == 4):
30         # if coloring is safe
31         if (isSafe(graph, color)):
32
33             # Print the solution
34             printSolution(color)
35             return True
36         return False
37
38     # Assign each color from 1 to m
39     for j in range(1, m + 1):
40         color[i] = j

```



```

41
42     # Recur of the rest vertices
43     if (graphColoring(graph, m, i + 1, color)):
44         return True
45     color[i] = 0
46     return False
47
48 # A utility function to print solution
49
50 def printSolution(color):
51     print("Solution Exists:" " Following are the assigned colors ")
52     for i in range(4):
53         print(color[i], end=" ")
54
55     print("\n")
56
57 # Driver code
58
59 if __name__ == '__main__':
60
61     graph = [
62         [0, 1, 1, 1],
63         [1, 0, 1, 0],
64         [1, 1, 0, 1],
65         [1, 0, 1, 0],
66     ]
67
68     m = 3          # Number of colors
69
70     # Initialize all color values as 0.
71     # This initialization is needed
72     # correct functioning of isSafe()
73     color = [0 for i in range(4)]
74
75     # Function call
76     if (not graphColoring(graph, m, 0, color)):
77         print("Solution does not exist")

```

Listing 7: Implementation of Graph Colouring Using BackTracking in Python

```

1 def greet(bot_name, birth_year):
2     print("Hello! My name is {0}.".format(bot_name))
3     print("I was created in {0}.".format(birth_year))
4
5 def remind_name():
6     print('\nPlease, remind me your name.')
7     name = input()
8     print("What a great name you have, {0}!".format(name))
9
10 def guess_age():
11     print('\nLet me guess your age.')
12     print('Enter remainders of dividing your age by 3, 5 and 7.')
13
14     rem3 = int(input())
15     rem5 = int(input())
16     rem7 = int(input())
17     age = (rem3 * 70 + rem5 * 21 + rem7 * 15) % 105
18
19     print("Your age is {0}; that's a good time to start programming!".format(age))
20
21 def number_guess():
22     import random
23     import math
24
25     print("\nHey! Here's a number guessing game for you!")
26
27     lower = int(input("\nEnter Lower bound:- "))
28     upper = int(input("Enter Upper bound:- "))
29     x = random.randint(lower, upper)
30
31     print("\n\tYou've only ",
32           round(math.log(upper - lower + 1, 2)),
33           " chances to guess the integer!\n")
34
35     count = 0
36     while count < math.log(upper - lower + 1, 2):
37         count += 1
38
39         guess = int(input("Guess a number:- "))
40

```

```

41     if x == guess:
42         print("Congratulations you did it in ",
43             count, " try")
44         break
45
46     elif x > guess:
47         print("You guessed too small!")
48
49     elif x < guess:
50         print("You Guessed too high!")
51
52 if count >= math.log(upper - lower + 1, 2):
53     print("\nThe number is %d" % x)
54     print("\tBetter Luck Next time!")
55
56 def count():
57     print('\nNow I will prove to you that I can count to any number you want.')
58     num = int(input())
59
60     counter = 0
61
62     while counter <= num:
63         print("{0} !".format(counter))
64         counter += 1
65
66 def test():
67     print("\nLet's test your programming knowledge.")
68     print("Why do we use methods?")
69     print("1. To repeat a statement multiple times.")
70     print("2. To decompose a program into several small subroutines.")
71     print("3. To determine the execution time of a program.")
72     print("4. To interrupt the execution of a program.")
73
74     answer = 2
75     guess = int(input())
76
77     while guess != answer:
78         print("Please, try again.")
79         guess = int(input())
80

```

```

81     print('Completed, have a nice day!')
82     print('.....')
83     print('.....')
84     print('.....')
85
86
87 def end():
88     print('Congratulations, have a nice day!')
89     print('.....')
90     print('.....')
91     print('.....')
92
93     input()
94
95 greet('ChatBot-v4', '2023')
96 remind_name()
97 guess_age()
98 number_guess()
99 count()
100 test()
101 end()

```

Listing 8: Develop an elementary catboat for any suitable customer interaction application in Python