
ΠΡΩΤΗ ΕΡΓΑΣΙΑ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής



Μάθημα: «ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΜΠΕΙΡΑ ΣΥΣΤΗΜΑΤΑ
(6ο εξ.)»

Ομάδα εργασίας:

Π18101 – ΑΝΑΣΤΑΣΙΑ ΙΩΑΝΝΑ ΜΕΞΑ
Π18123 – ΒΑΣΙΛΙΚΗ ΠΑΣΙΑ

Γενικές επισημάνσεις:

- Η εργασία που μας ανατέθηκε είναι η 4^η (επίλυση λαβυρίνθου).
- Η γλώσσα προγραμματισμού που χρησιμοποιήσαμε είναι η Python.
- Οι δύο αλγόριθμοι αναζήτησης που υλοποιήσαμε είναι οι:
 1. Αναζήτηση πρώτα κατά πλάτος
 2. Αναζήτηση πρώτα κατά βάθος
- Η διαμόρφωση του πίνακα για τις ανάγκες του κώδικά μας φαίνεται στο παρακάτω σχήμα. Αναλυτικότερα, ο counter j έχει εύρος τιμών από 0 έως n (όπου n η διάσταση του πίνακα) και μετράει τις γραμμές του πίνακα. Ενώ, ο counter i έχει εύρος τιμών από 0 μέχρι n και μετράει τις στήλες του πίνακα.

| | | | | | |
|---|---|---|---|---|---|
| j | | | | | |
| 0 | | | | | |
| 1 | | | | | |
| 2 | | | | | X |
| 3 | | | | | |
| 4 | S | | | | |
| | 0 | 1 | 2 | 3 | 4 |
| | i | | | | |

Ανάλυση λειτουργίας συναρτήσεων:

Έχουμε δημιουργήσει συνολικά 6 συναρτήσεις μέσα στον κώδικά μας. Συγκεκριμένα τις:

1. draw_maze()
2. draw_maze2()
3. find_start_points()
4. print_maze()
5. valid_move()
6. find_end()

Παρακάτω αναλύονται οι συναρτήσεις και οι λειτουργίες τους.

1. Η draw_maze() δημιουργεί τον πίνακα (λαβύρινθο) που το πρόγραμμα καλείται να επιλύσει. Το πρόγραμμά μας μπορεί να επιλύσει όποιον λαβύρινθο σχεδιάσουμε εμείς, αρκεί ο πίνακας να έχει διαστάσεις nxn. Ο χαρακτήρας «■» δηλώνει έναν «τοίχο», δηλαδή δεν μπορούμε να περάσουμε από το συγκεκριμένο κελί, ο χαρακτήρας « » δηλώνει ένα κενό κελί από το οποίο μπορούμε να περάσουμε, ο χαρακτήρας «S» δηλώνει το σημείο εκκίνησης, ενώ ο χαρακτήρας «X» δηλώνει το σημείο εξόδου. Σκοπός του προγράμματος είναι να βρει μια διαδρομή (ανάλογα με τον αλγόριθμο αναζήτησης) η οποία θα ξεκινά από το S και θα καταλήγει στο X.
2. Η draw_maze2() εκτελεί την ίδια λειτουργία με την draw_maze(), απλά την έχουμε ως ξεχωριστή συνάρτηση η οποία περιέχει ένα μεγαλύτερο παράδειγμα λαβυρίνθου, για να επιδείξουμε ότι λειτουργεί για nxn διαστάσεις.

Παρακάτω υπάρχει screenshot με τον κώδικα των 2 συναρτήσεων.

```

1 import queue
2
3 # creates the maze (2 examples)
4 # S is the starting point and X is the exit point
5 def draw_maze():
6     maze = []
7     maze.append(["■", "■", "■", "■", "■"])
8     maze.append(["■", " ", "■", " ", "■"])
9     maze.append(["■", " ", " ", " ", "X"])
10    maze.append(["■", " ", " ", "■", " "])
11    maze.append(["S", " ", "■", "■", "■"])
12
13    return maze
14
15
16 def draw_maze2():
17     maze = []
18     maze.append(["■", "■", "■", "■", "■", "■", "■", "X", "■"])
19     maze.append(["■", " ", " ", " ", " ", " ", " ", " ", "■"])
20     maze.append(["■", "■", "■", "■", "■", "■", "■", " ", "■"])
21     maze.append(["■", " ", " ", " ", " ", " ", " ", " ", "■"])
22     maze.append(["■", " ", "■", " ", " ", " ", " ", " ", "■"])
23     maze.append(["■", " ", "■", " ", " ", " ", " ", " ", "■"])
24     maze.append(["■", " ", " ", " ", " ", " ", " ", " ", "■"])
25     maze.append(["■", " ", " ", " ", " ", " ", " ", " ", "■"])
26     maze.append(["S", " ", " ", " ", " ", " ", " ", " ", "■"])
27
28    return maze
29

```

3. Η find_start_points() βρίσκει τις συντεταγμένες του κελιού που βρίσκεται ο χαρακτήρας «S».

```

30 def find_start_points(maze): # finds the coordinates of the starting point
31     for l in range(0, len(maze)): # for every row
32         for x, pos in enumerate(maze[l]): # for every column
33             #print(x, pos)
34             if pos == "S":
35                 i = x # column
36                 j = l # row
37                 return i, j
38

```

4. Η print_maze() εκτυπώνει τον λαβύρινθο μαζί με την σημειωμένη τελική διαδρομή καθώς και τις κινήσεις που ακολούθησε.

```

39 def print_maze(maze, path=""): # printing the final maze with solution
40     i, j = find_start_points(maze)
41     pos = set()
42     for move in path:
43         if move == "L":
44             i -= 1
45         elif move == "R":
46             i += 1
47         elif move == "U":
48             j -= 1
49         elif move == "D":
50             j += 1
51         pos.add((j, i))
52
53     for j, row in enumerate(maze):
54         for i, col in enumerate(row):
55             if (j, i) in pos: # for every point in the solution path
56                 print("X ", end="") # mark
57
58             else:
59                 print(col + " ", end="")
60
61     print()
62
63

```

Ενδεικτικό αποτέλεσμα της συνάρτησης:

```
stacy@stacy-VirtualBox:~$ python3 test.py
1. Breadth First Search
2. Depth First Search

Enter your choice: 1

Found: RURURR
  █ █ █ █ █
  █   █   █
  █   x x x
  █ x x █
  S x █ █ █
```

5. Η `valid_move()` ελέγχει αν η κίνηση που επρόκειτο να πραγματοποιηθεί είναι έγκυρη.

```
64 def valid_move(maze, moves): # check if the move is valid
65     i, j = find_start_points(maze)
66     for move in moves:
67         if move == "L":
68             i -= 1
69
70         elif move == "R":
71             i += 1
72
73         elif move == "U":
74             j -= 1
75
76         elif move == "D":
77             j += 1
78     #print(i, j)
79     if not(0 <= i < len(maze) and 0 <= j < len(maze)): # outside the borders of the maze
80         #print("false")
81         return False
82     elif (maze[j][i] == "█"): # wall
83         #print("false")
84         return False
85     #print("true")
86     return True
87
88
```

6. Η `find_end()` βρίσκει τις συντεταγμένες του κελιού που βρίσκεται ο χαρακτήρας «X».

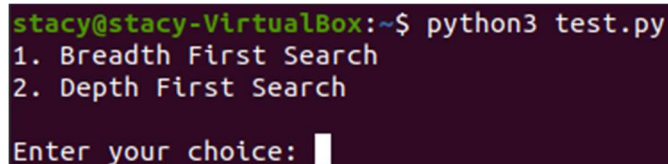
```
89 def find_end(maze, moves): # finds the coordinates of the exit point
90     i, j = find_start_points(maze)
91     for move in moves:
92         if move == "L":
93             i -= 1
94
95         elif move == "R":
96             i += 1
97
98         elif move == "U":
99             j -= 1
100
101         elif move == "D":
102             j += 1
103
104     if maze[j][i] == "X": # if we reached the exit point, print solution
105         print("\nFound: " + moves)
106         print_maze(maze, moves)
107         return True
108
109     return False
110
111
```

Επεξήγηση κώδικα:

Το κύριο κομμάτι του προγράμματος:

```
112 combinations = queue.Queue()
113 combinations.put("")
114 lst = ""
115 maze = draw_maze()
116 print("1. Breadth First Search")
117 print("2. Depth First Search\n")
118
119 # cheking user input
120 while True:
121     try:
122         inp = int(input("Enter your choice: "))
123         if inp < 1 or inp > 2:
124             raise ValueError
125         break
126     except ValueError:
127         print("Please enter a valid choice\n")
128
129
130 while not find_end(maze, lst): # run until we reach the exit point
131     lst = combinations.get()
132     #print(lst)
133     if inp == 1:
134         s = ["L", "R", "U", "D"] # Breadth First Search
135     else:
136         s = ["U", "R", "D", "L"] # Depth First Search
137     for j in s: # for its move (left, right, etc...)
138         put = lst + j
139         if valid_move(maze, put): # if move is valid
140             combinations.put(put) # remember the valid move
```

Αρχικά ορίζουμε το queue combinations στο οποίο αργότερα θα αποθηκεύονται οι έγκυρες κινήσεις και ο συνδυασμός τους. Έπειτα, δίνεται η δυνατότητα στον χρήστη να επιλέξει με ποιόν αλγόριθμο αναζήτησης θέλει να τρέξει το πρόγραμμα και ελέγχεται το input του.



```
stacy@stacy-VirtualBox:~$ python3 test.py
1. Breadth First Search
2. Depth First Search

Enter your choice: █
```

Συνεχίζοντας, το while loop επαναλαμβάνεται όσο η συνάρτηση find_end() δεν έχει βρει το σημείο εξόδου, δηλαδή δεν έχουμε βρει μια διαδρομή η οποία να καταλήγει στο κελί με τον χαρακτήρα «X». Στις ήδη υπάρχουσες διαδρομές, προσθέτουμε κάθε φορά μια πιθανή κίνηση (Αριστερά, Δεξιά, Πάνω, Κάτω για τον αλγόριθμο αναζήτησης πρώτα κατά πλάτος και Πάνω, Δεξιά, Κάτω, Αριστερά για τον αλγόριθμο αναζήτησης πρώτα κατά βάθος) και ελέγχουμε την εγκυρότητα της. Αν είναι μια έγκυρη κίνηση την προσθέτουμε στο queue και η διαδικασία επαναλαμβάνεται, αλλιώς την απορρίπτουμε, δεν μπαίνει στο queue και συνεχίζουμε στην επόμενη κίνηση.

Παραδείγματα εξόδου του κώδικα:

```
stacy@stacy-VirtualBox:~$ python3 test.py
```

1. Breadth First Search

2. Depth First Search

```
Enter your choice: 1
```

Found: RURRRRUUURRUUUU

```
stacy@stacy-VirtualBox:~$ python3 test.py
```

1. Breadth First Search

2. Depth First Search

Enter your choice: 2

Found: RURRUUUUUURRRRU

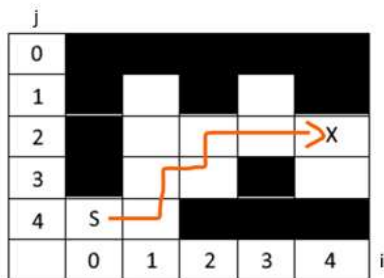
A 10x10 grid representing a 1000m x 1000m area. The grid is mostly empty, with a few 'X' marks indicating specific locations. The 'X' marks are located at (row, column) coordinates: (1, 9), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (3, 4), (4, 4), (5, 4), (6, 4), (7, 4), (8, 1), (8, 2), (8, 3), (8, 4), (9, 1). The bottom-left corner is labeled 'S'.

Παρακάτω ακολουθεί ένα επεξηγηματικό δικό μας σχέδιο για την λειτουργία του προγράμματος. Συγκεκριμένα, δείχνει πως γίνεται η διαδικασία ελέγχου εγκυρότητας για την τελική διαδρομή με την χρήση και των δύο αλγορίθμων αναζήτησης.

Παράδειγμα υλοποίησης λαβυρίνθου

Τρίτη, 16 Μαρτίου 2021 3:53 μμ

Λύση με τον αλγόριθμο Breadth First Search



| State | i | j | maze[i][j] |
|-------|---|---|------------|
| | 0 | 4 | S |
| R | 1 | 4 | ✓ |
| U | 1 | 3 | ✓ |
| R | 2 | 3 | ✓ |
| U | 2 | 2 | ✓ |
| R | 3 | 2 | ✓ |
| R | 4 | 2 | ✗ |

```
if move == "L":
    i -= 1
```

```
elif move == "R":
    i += 1
```

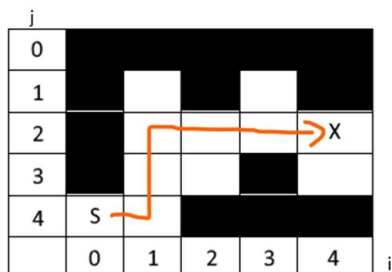
```
elif move == "U":
    j -= 1
```

```
elif move == "D":
    j += 1
```

Παράδειγμα υλοποίησης λαβυρίνθου

Τρίτη, 16 Μαρτίου 2021 3:53 μμ

Λύση με τον αλγόριθμο Depth First Search



| State | i | j | maze[i][j] |
|-------|---|---|------------|
| | 0 | 4 | S |
| R | 1 | 4 | ✓ |
| U | 1 | 3 | ✓ |
| U | 1 | 2 | ✓ |
| R | 2 | 2 | ✓ |
| R | 3 | 2 | ✓ |
| R | 4 | 2 | ✗ |

```
if move == "L":
    i -= 1
```

```
elif move == "R":
    i += 1
```

```
elif move == "U":
    j -= 1
```

```
elif move == "D":
    j += 1
```

Πηγαίος κώδικας:

```
import queue
```

```
# creates the maze (2 examples)
```

```
# S is the starting point and X is the exit point
```

```
def draw_maze():
```

```
    maze = []
```

```
    maze.append(["■","■", "■", "■", "■"])
```

```
    maze.append(["■", " ", "■", " ", "■"])
```

```
    maze.append(["■", " ", " ", " ", "X"])
```

```
    maze.append(["■", " ", " ", "■", " "])
```

```
    maze.append(["S", " ", "■", "■", "■"])
```

```
    return maze
```

```
def draw_maze2():
```

```
    maze = []
```

```
    maze.append(["■","■", "■", "■", "■", "■", "■", "X", "■"])
```

```
    maze.append(["■", " ", " ", " ", " ", " ", " ", " ", "■"])
```

```
    maze.append(["■","■", "■", " ", "■", "■", "■", " ", "■"])
```

```
    maze.append(["■", " ", "■", " ", "■", " ", "■", " ", "■"])
```

```
    maze.append(["■", " ", "■", " ", "■", " ", " ", " ", "■"])
```

```
    maze.append(["■", " ", "■", " ", "■", " ", "■", " ", "■"])
```

```
    maze.append(["■", " ", "■", " ", "■", " ", "■", " ", "■"])
```

```
    maze.append(["■", " ", " ", " ", " ", " ", "■", " ", "■"])
```

```
    maze.append(["S", " ", "■", "■", "■", "■", "■", "■", "■"])
```

```
    return maze
```

```
def find_start_points(maze):      # finds the coordinates of the starting point
```

```
    for l in range(0,len(maze)):  # for every row
```

```
        for x, pos in enumerate(maze[l]): # for every column
```

```
            #print(x, pos)
```

```
            if pos == "S":
```

```
                i = x      # column
```



```
j = l      # row
```

```
return i, j
```

```
def print_maze(maze, path=""):    # printing the final maze with solution
```

```
    i, j = find_start_points(maze)
```

```
    pos = set()
```

```
    for move in path:
```

```
        if move == "L":
```

```
            i -= 1
```

```
        elif move == "R":
```

```
            i += 1
```

```
        elif move == "U":
```

```
            j -= 1
```

```
        elif move == "D":
```

```
            j += 1
```

```
        pos.add((j, i))
```

```
    for j, row in enumerate(maze):
```

```
        for i, col in enumerate(row):
```

```
            if (j, i) in pos:    # for every point in the solution path
```

```
                print("x ", end="")    # mark
```

```
            else:
```

```
                print(col + " ", end="")
```

```
        print()
```

```
def valid_move(maze, moves):    # check if the move is valid
```

```
    i, j = find_start_points(maze)
```

```
    for move in moves:
```

```
        if move == "L":
```

```
            i -= 1
```

```
        elif move == "R":
```

```
            i += 1
```

```

elif move == "U":
    j -= 1

elif move == "D":
    j += 1
#print(i, j)
if not(0 <= i < len(maze) and 0 <= j < len(maze)): # outside the borders of the maze
    #print("false")
    return False
elif (maze[j][i] == "■"): # wall
    #print("false")
    return False
#print("true")
return True

```

```

def find_end(maze, moves):      # finds the coordinates of the exit point
    i, j = find_start_points(maze)
    for move in moves:
        if move == "L":
            i -= 1

        elif move == "R":
            i += 1

        elif move == "U":
            j -= 1

        elif move == "D":
            j += 1

    if maze[j][i] == "X":      # if we reached the exit point, print solution
        print("\nFound: " + moves)
        print_maze(maze, moves)
        return True

```

```
return False
```

```
combinations = queue.Queue()
```

```
combinations.put("")
```

```
lst = ""
```

```
maze = draw_maze()
```

```
print("1. Breadth First Search")
```

```
print("2. Depth First Search\n")
```

```
# cheking user input
```

```
while True:
```

```
    try:
```

```
        inp = int(input("Enter your choice: "))
```

```
        if inp < 1 or inp > 2:
```

```
            raise ValueError
```

```
        break
```

```
    except ValueError:
```

```
        print("Please enter a valid choice\n")
```

```
while not find_end(maze, lst): # run until we reach the exit point
```

```
    lst = combinations.get()
```

```
    #print(lst)
```

```
    if inp == 1:
```

```
        s = ["L", "R", "U", "D"] # Breadth First Search
```

```
    else:
```

```
        s = ["U", "R", "D", "L"] # Depth First Search
```

```
    for j in s: # for its move (left, right, etc...)
```

```
        put = lst + j
```

```
        if valid_move(maze, put): # if move is valid
```

```
            combinations.put(put)    # remember the valid move
```