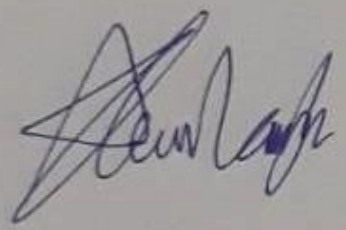
Python Path Finding with Breadth First Search

BINTANG NURALAMSYAH / 05111740000002

TRIA NUR AISYAH AMINI / 05111740000092

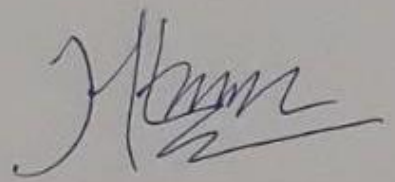
"By the name of Allah (God) Almighty, herewith we pledge and truly declare that we have solved quiz 2 by ourselves, didn't do any cheating by any means, didn't do any plagiarism, and didn't accept anybody's help by any means. We are going to accept all of the consequences by any means if it has proven that we have been done any cheating and/or plagiarism."

Surabaya, date, 30 March 2019



Bintang Nuralamsyah

05111740000002



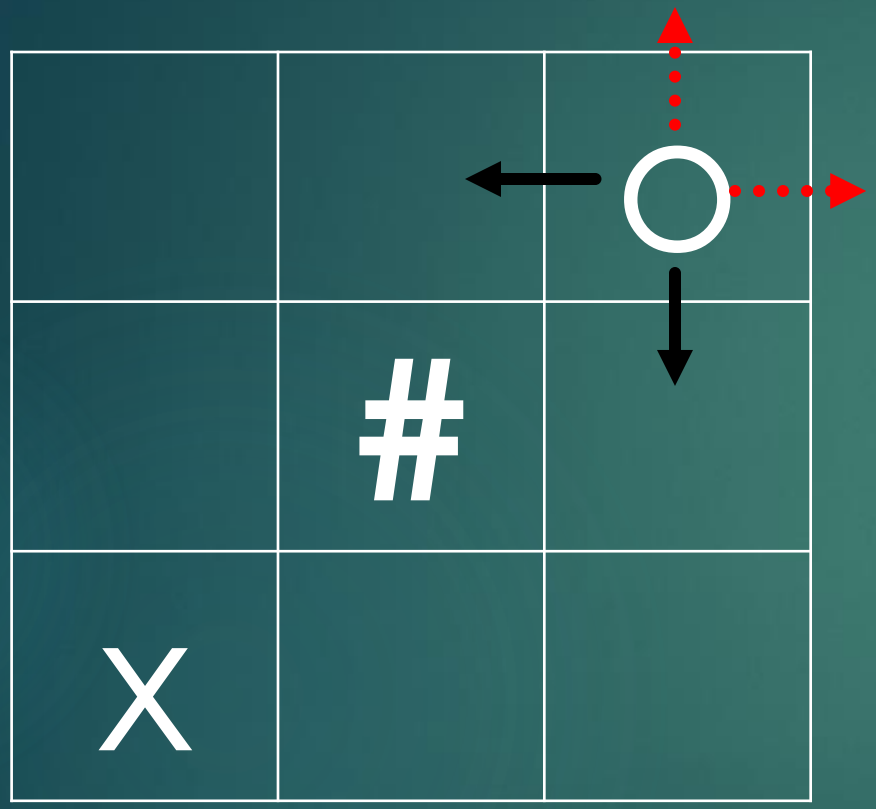
Tria Nur Aisyah Amini

05111740000092

Theorem and Abstraction

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.
- It uses the opposite strategy as depth-first search, which instead explores the highest-depth nodes first before being forced to backtrack and expand shallower nodes.
- BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse, in his (rejected) Ph.D. thesis on the Plankalkül programming language, but this was not published until 1972. It was reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze and later developed by C. Y. Lee into a wire routing algorithm (published 1961).
- Source : https://en.wikipedia.org/wiki/Breadth-first_search

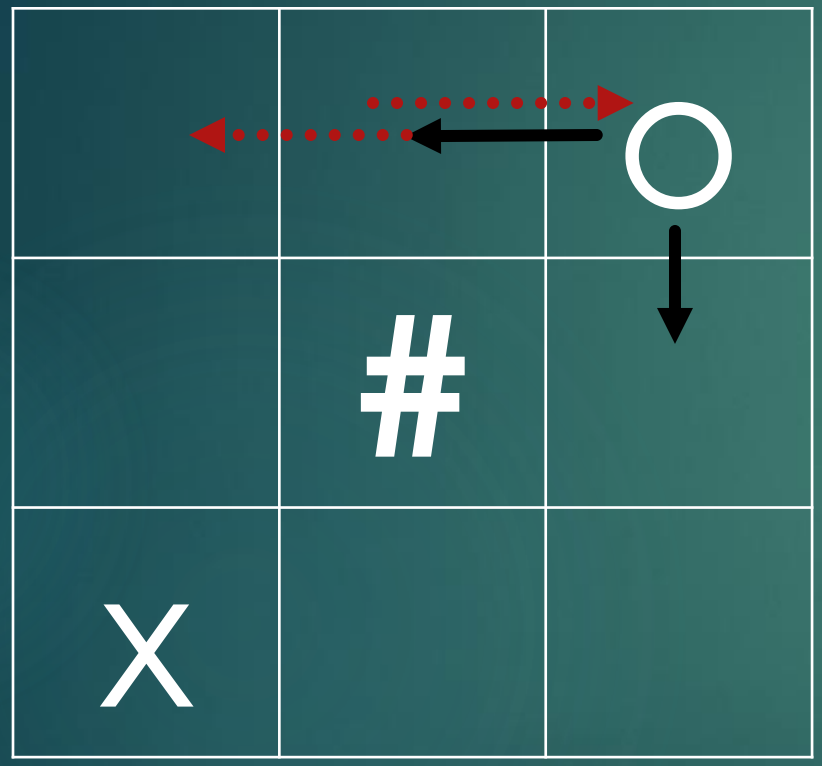
Explanation and Analysis



For example we drawing 3x3 grid, and let we say # is an obstacle, ‘o’ is starting point and ‘x’ is the goal. What we’ll do is create all the possibility moves.

First move , the valid moves we can do is we go to the left or we go down. Because if we try to go right or up , those are not valid moves so we’re not gonna add them to the queue . So when we’re checking if we reach this destination and we’re only checking the valid moves there’s no point to checking moves outside of the range

Explanation and Analysis



So we have two possible moves, left and down. So in our queue we're gonna add left and down :

[L,D]

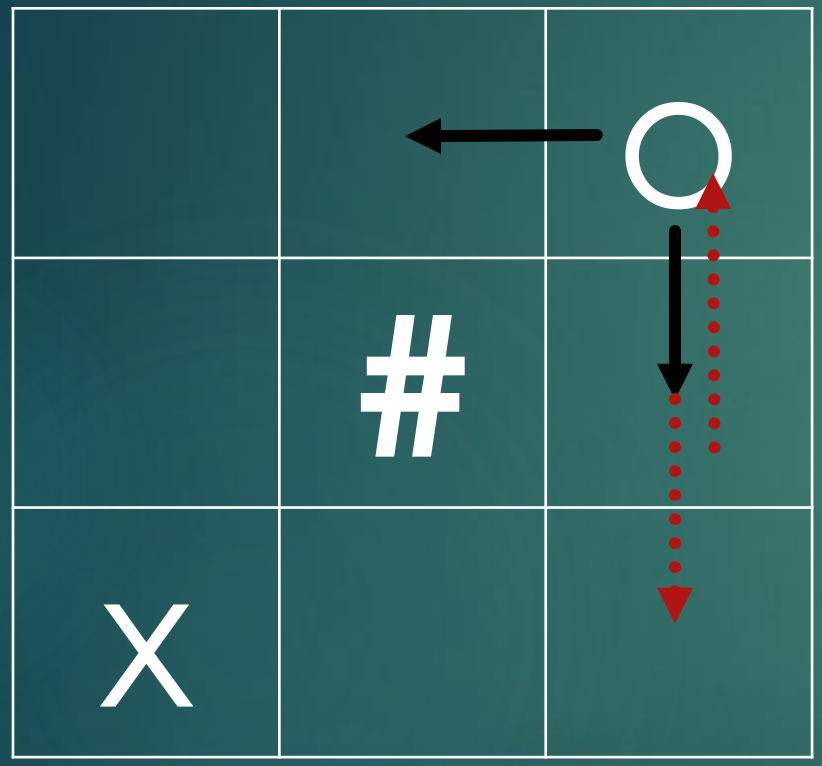
So now, we repeat the process, we'll dequeue L (left),

[D]

Now we're at L,look at the blue arrow, we're looking at this move. We can go right or left, so two moves we're gonna add to queueu

[D, LL,LR]

Explanation and Analysis



And then look at down moves, we'll dequeue down

[LL,LR]

The possible moves is we can go down or up, so two moves we're gonna add to queue

[LL,LR,DD,DU]

Just continue repeating this process and we're gonna get a move that's either equal to **[LLDDD]** or **[DDLLL]** each of this are only five moves, so either path is valid to go and that how we're gonna generate the path to find our point.

Source Code Analysis

Program Documentation

- Our Program Purpose is to help user to find path in a labyrinth / maze.
- We use 1 main function and 6 support function (buatLabirin, poscount, validMove, endPoint, changePath, and printMap)
- To make the model of the labyrinth,
  open the source code then edit the labyrinth inside the buatlabirin function.
- The output of the program will be steps the should be taken and the map with th e path or "No. Valid path"
- S means endpoint and M means startpoint

1. buatLabirin()

```
68  #Membuat labirin
69  def buatLabirin():
70      labirin = []
71      labirin.append(["#", "#", " ", "#"])
72      labirin.append(["#", "M", " ", "#"])
73      labirin.append(["#", "#", " ", "#"])
74      labirin.append(["#", "#", " ", "#"])
75      labirin.append(["#", "#", " ", "#"])
76      labirin.append(["#", "#", "S", "#"])
77
78      return labirin
79
```

We use this function to model our labyrinth. The datatype of the labyrinth is list []. List[] is a python datatype collection that allows duplicated member.

Poscount(labirin)

- Function that will calculate the maximal steps that could be taken. It counts the number of white space plus M (start point) and S (end point)

```
116     def poscount(labirin):
117         steps = 0
118         for j, row in enumerate(labirin):
119             for i,col in enumerate(row):
120                 if col == " ":
121                     steps +=1
122         steps+=3
123         return steps
124
```

ValidMove(labirin, langkah)

The purpose of this function is to check whether a move that is just taken is valid or not. Valid means the present move is not outside the maze and not on the "#" (obstacle) sign. If the move is valid, the function will return true, but if the move is invalid, the function will return false.

```python
import queue

#Menentukan sebuah pilihan langkah valid / tidak
def validMove(labirin, langkah):
    for j, row in enumerate(labirin):
        for i,col in enumerate(row):
            if col == "M":
                x1 = i
                y1 = j
                break

    x = x1
    y = y1
    #print(x)
    #print(y)

    for i in langkah:
        if i == "U":
            y -= 1
        elif i == "D":
            y += 1
        elif i == "L":
            x -= 1
        elif i == "R":
            x += 1

    if not ( 0<= x < len(labirin[0]) and 0 <= y < len(labirin)):
        #print("FALSE")
        return False
    elif labirin[y][x] == "#":
        #print("FALSE")
```

changePath(labirin,langkah)

- ChangePath function is a function that gives + sign to some points at the labyrinth as the symbol that those points are parts of the path.

```
80  #Memberikan tanda + pada jalan yang dipilih
81  def changePath(labirin, langkah):
82      for j, row in enumerate(labirin):
83          for i,col in enumerate(row):
84              if col == "M":
85                  x1 = i
86                  y1 = j
87                  break
88
89      x = x1
90      y = y1
91      #print(x)
92      #print(y)
93
94      for i in langkah:
95          if i == "U":
96              y -= 1
97          elif i == "D":
98              y += 1
99          elif i == "L":
100             x -= 1
101         elif i == "R":
102             x += 1
103         if labirin[y][x] != "S":
104             labirin[y][x] = "+"
105
106
107     return labirin
108
```

EndPoint(labirin,langkah)

- This function purpose is to check whether the present moves collection has reached the destination or not. If the present collection of moves has reached the destination, the function will print the moves collection and call changePath and printMap function.

```
37  #Menentukan apakah sudah mencapai titik akhir (S)
38  def endPoint(labirin, langkah):
39      for j, row in enumerate(labirin):
40          for i,col in enumerate(row):
41              if col == "M":
42                  x1 = i
43                  y1 = j
44                  break
45
46      x = x1
47      y = y1
48      #print(x)
49      #print(y)
50
51      for i in langkah:
52          if i == "U":
53              y -= 1
54          elif i == "D":
55              y += 1
56          elif i == "L":
57              x -= 1
58          elif i == "R":
59              x += 1
60      if labirin[y][x] == "S":
61          changePath(labirin,langkah)
62          print("langkah yang didapat " + langkah)
63          printMap(labirin)
64          return True
65      #print("Tidak sampai")
66      return False
67
```

PrintMap(labirin)

- With this function we can print the labyrinth model with the path that is choosen.

```
109     #Mencetak labirin beserta path yang dipilih
110     def printMap(labirin):
111         for j, row in enumerate(labirin):
112             for i, col in enumerate(row):
113                 print(col + " ",end="")
114             print()
115
```

Main

- This is the main function of our program. in this function we create the queue (FIFO) that will collect possible moves from the BFS. We also create a variabel that will contain the steps that we took.
- The logic of the function :
- While the present move has not reached the destination (letter S) POP the queue, then add it to present move variable

```
127     #1. Membuat labirin
128     labirin = buatLabirin()
129     #2. Membuat Queue untuk BFS
130     bfs = queue.Queue()
131     bfs.put("")
132     #3. Membuat variabel untuk menyimpan
133     #kemungkinan langkah
134
135     posstep = ""
136     totstep = poscount(labirin)
137     print(totstep)
138
139     #Proses pencarian Jalur
140     while not endPoint(labirin,posstep):
141         if len(posstep) > totstep:
142             print("no Valid Path")
143             exit(2)
144
145         posstep = bfs.get()
146         for direction in ["U","D","L","R"]:
147             choice = posstep + direction
148             if validMove(labirin,choice):
149                 bfs.put(choice)
150
```

Main

- List the next possible moves (choice variable contains the next possible moves)
- If the next possible moves is valid, push it to the queue
- Repeat this logic until we found the path
- If there is no valid path (when the quantity of steps that have been taken is greater than the number of vertex, (in this case the white space element + S>destination +M>start point)) the program will print " no valid path and exit with exit code (2)

```
127     #1. Membuat labirin
128     labirin = buatLabirin()
129     #2. Membuat Queue untuk BFS
130     bfs = queue.Queue()
131     bfs.put("")
132     #3. Membuat variabel untuk menyimpan
133     #kemungkinan langkah
134
135     posstep = ""
136     totstep = poscount(labirin)
137     print(totstep)
138
139     #Proses pencarian Jalur
140     while not endPoint(labirin,posstep):
141         if len(posstep) > totstep:
142             print("no Valid Path")
143             exit(2)
144
145         posstep = bfs.get()
146         for direction in ["U","D","L","R"]:
147             choice = posstep + direction
148             if validMove(labirin,choice):
149                 bfs.put(choice)
150
```

Example 1 (labyrinth that has valid path)

From the result, we can see that we need 7 steps to reach the destination (S). and the steps are down > down > down > down > left > left > up

```
labirin.append(["#", "#", "M", "#"])
labirin.append(["#", " ", " ", "#"])
labirin.append([" ", "#", " ", "#"])
labirin.append(["S", "#", " ", "#"])
labirin.append([" ", " ", " ", "#"])
labirin.append(["#", "#", "#", "#"])
```

```
langkah yang didapat DDDDLLU
# # M #
#   + #
  # + #
S # + #
+ + + #
# # # #

Process finished with exit code 0
```

Example 2 (Labyrinth without valid path)

```
71      labirin.append(["#", "#", "M", "#"])
72      labirin.append(["#", " ", " ", "#"])
73      labirin.append([" ", "#", " ", "#"])
74      labirin.append(["S", "#", " ", "#"])
75      labirin.append([" ", "#", " ", "#"])
76      labirin.append(["#", "#", "#", "#"])
77
```

```
/mnt/510c764a-8824-48a9-89b5-6c8791f8
no Valid Path

Process finished with exit code 2
```

- From the labyrinth we can know if there is no valid path to the destination
- The result of the program is no valid path because the number of steps that has been taken is greater than the maximal step that could be taken.