MAZESOLVER USING BFS

LAB REPORT

Submitted by

Shaurya Singh Srinet (RA2111032010006)

Under the Guidance of

Dr. A Prabhu Chakkaravarthy

Assistant Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

with specialization in Internet of Things

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

MAY 2023

COLLEGE OF ENGINEERING & TECHNOLOGY SRM INSTITUTE OF SCIENCE & TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUE – 603 203

Chengalpattu District

BONAFIDE CERTIFICATE

Register No. RA2111032010006 Certified to be the bonafide work done by Shaurya Singh Srinet of II Year/IV Sem B. Tech Degree Course in the **Practical Course – 18CSC204J - Design and Analysis of Algorithms** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY,** Kattankulathur during the academic year 2022 – 2023.

SIGNATURE

Faculty In-Charge
Dr. A Prabhu Chakkaravarthy
Assistant Professor
Department of Networking and Communications
SRM Institute of Science and Technology

SIGNATURE

HEAD OF THE DEPARTMENT
Dr. Annapurani Panaiyappan. K
Professor and Head,
Department of Networking and Communications
SRM Institute of Science and Technology

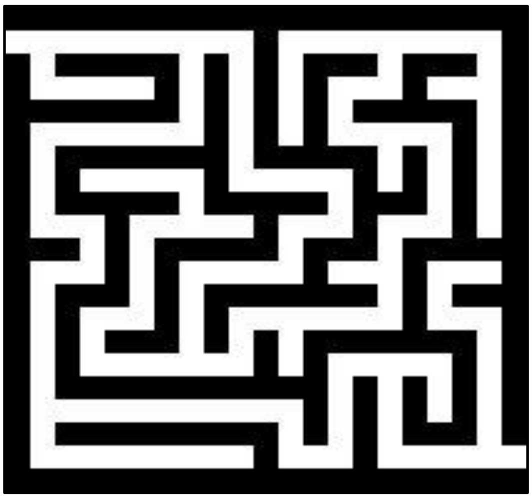
TABLE OF CONTENTS

PROBLEM STATEMENT

This program takes a gif of a maze as input and outputs a solution to the maze. It converts the gif into a matrix where each cell represents a 5x5 area of the original gif. It then uses a breadth-first search algorithm to find the shortest path from the starting cell to the ending cell and displays the solution as a gif.

PROBLEM EXPLANATION

The objective of the maze solver using BFS is to develop an algorithm that can find the shortest path through a given maze using Breadth First Search (BFS) traversal. The algorithm should be able to take a maze as input, and then apply BFS to explore the maze until the end point is reached. The final output should be the shortest path from the start point to the end point, along with the steps taken to reach the end point. This project can be useful in areas such as robotics, gaming, and navigation. By finding the shortest path through a maze, robots can navigate through obstacles more efficiently. In gaming, this algorithm can be used to guide characters through a game level. In navigation, the algorithm can be used to find the shortest route between two points on a map. Overall, the maze solver using BFS algorithm has many practical applications in various fields.



```
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1],
[1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
[1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1],
[1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1],
[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1],
[1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1],
[1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
[1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1],
[1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
[1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1],
[1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
[1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1],
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1],
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

ALGORITHM

Step 1: Read the input maze and identify the start and end points.

Step 2: Create an empty queue and add the start point to it.

Step 3: Initialize a visited set to keep track of already visited points and mark the start point as visited.

Step 4: Loop until the queue is empty:

a. Dequeue the front point from the queue.

b. If the dequeued point is the end point, then the algorithm has found the shortest path.

c. Otherwise, enqueue all adjacent unvisited points to the queue and mark them as visited.

Step 5: If the end point is reached, trace back the path from the end point to the start point by following the parents of each point.

Step 6: Output the shortest path from the start point to the end point along with the steps taken to reach the end point.

This algorithm implements BFS traversal to explore the maze, finding the shortest path from start to end. The visited set is used to avoid revisiting previously explored points. Finally, the algo-rithm backtracks from the end point to the start point to determine the shortest path.

SOURCE CODE

```
def ConvertImage(ImageName):
    from PIL import Image
    import numpy as np

    # Open the maze image and make greyscale, and get its dimensions
    im = Image.open(ImageName).convert('L')
    #im.show()
    w, h = im.size

    # Ensure all black pixels are 0 and all white pixels are 1
    binary = im.point(lambda p: p < 128 and 1)

    # Resize to half its height and width so we can fit on Stack Overflow, get new dimensions
    binary = binary.resize((w//2,h//2),Image.NEAREST)
    w, h = binary.size

    # Convert to Numpy array - because that's how images are best stored and processed in Py-
thon
    nim = np.array(binary)

    # Each cell of the maze is represented by 5 numbers. Therefore change scaling FROM (5
number: 1 cell) TO (1 numer: 1 cell)
    # initialize maze matrix
    maze = [[0 for i in range(int(w/5))] for j in range(int(h/5))]

    # go through every 5th number in each row and add it sequencially to the new matrix (maze).
    ri = ci = 0
    r = c = 4
    while ri < h/5:
        ci = 0
```

```
        c = 4
        while ci < w/5:
            maze[ri][ci] = nim[r][c]
            # print(maze[ri][ci], end='')       # for testing purpose print final maze matrix
            ci += 1
            c += 5
        # print()
        ri += 1
        r += 5
    # print("\n\n")
    return [maze,int(h/5),int(w/5)]
    for r in range(4, h, 5):
        for c in range(4, w, 5):
            print(nim[r,c],end='')
        print()

ConvertImage('maze.png')
#for solving maze
from Image2Array_CustomLibrary import *
from PIL import Image, ImageDraw
images = []
maze_name = input("Enter Image name with file format (eg. 'maze.png'): ")
output_name = input("Save output as (filename alone): ")
print("PLEAS EGIVE ME SOME TIME....")
maze_loc = './inputs/' + maze_name
a, rows, columns = ConvertImage(maze_loc)
zoom = 20
borders = 5
start = 1,0
end = rows-2,columns-1
```

```
# print(a,rows, columns)
def make_step(k):
  for i in range(len(m)):
    for j in range(len(m[i])):
      if m[i][j] == k:
        if i>0 and m[i-1][j] == 0 and a[i-1][j] == 0:
          m[i-1][j] = k + 1
        if j>0 and m[i][j-1] == 0 and a[i][j-1] == 0:
          m[i][j-1] = k + 1
        if i<len(m)-1 and m[i+1][j] == 0 and a[i+1][j] == 0:
          m[i+1][j] = k + 1
        if j<len(m[i])-1 and m[i][j+1] == 0 and a[i][j+1] == 0:
           m[i][j+1] = k + 1
def print_m(m):
    for i in range(len(m)):
        for j in range(len(m[i])):
            print( str(m[i][j]).ljust(2),end=' ')
        print()
def draw_matrix(a,m, the_path = []):
    im = Image.new('RGB', (zoom * len(a[0]), zoom * len(a)), (255, 255, 255))
    draw = ImageDraw.Draw(im)
    for i in range(len(a)):
        for j in range(len(a[i])):
            color = (255, 255, 255)
            r = 0
            if a[i][j] == 1:
                color = (0, 0, 0)
            if i == start[0] and j == start[1]:
                color = (0, 255, 0)
                r = borders
            if i == end[0] and j == end[1]:
```

```
            color = (0, 255, 0)
            r = borders
        draw.rectangle((j*zoom+r, i*zoom+r, j*zoom+zoom-r-1, i*zoom+zoom-r-1), fill=color)
        if m[i][j] > 0:
            r = borders
            draw.ellipse((j * zoom + r, i * zoom + r, j * zoom + zoom - r - 1, i * zoom + zoom - r - 1),
                           fill=(255,0,0))
    for u in range(len(the_path)-1):
        y = the_path[u][0]*zoom + int(zoom/2)
        x = the_path[u][1]*zoom + int(zoom/2)
        y1 = the_path[u+1][0]*zoom + int(zoom/2)
        x1 = the_path[u+1][1]*zoom + int(zoom/2)
        draw.line((x,y,x1,y1), fill=(255, 0,0), width=5)
    draw.rectangle((0, 0, zoom * len(a[0]), zoom * len(a)), outline=(0,255,0), width=2)
    images.append(im)
m = []
for i in range(rows):
    m.append([])
    for j in range(columns):
        m[-1].append(0)
i,j = start
m[i][j] = 1
k = 0
while m[end[0]][end[1]] == 0:
    k += 1
    make_step(k)
    draw_matrix(a, m)
i, j = end
k = m[i][j]
the_path = [(i,j)]
```

```
while k > 1:
  if i > 0 and m[i - 1][j] == k-1:
    i, j = i-1, j
    the_path.append((i, j))
    k-=1
  elif j > 0 and m[i][j - 1] == k-1:
    i, j = i, j-1
    the_path.append((i, j))
    k-=1
  elif i < len(m) - 1 and m[i + 1][j] == k-1:
    i, j = i+1, j
    the_path.append((i, j))
    k-=1
  elif j < len(m[i]) - 1 and m[i][j + 1] == k-1:
    i, j = i, j+1
    the_path.append((i, j))
    k -= 1
  draw_matrix(a, m, the_path)
for i in range(10):
    if i % 2 == 0:
        draw_matrix(a, m, the_path)
    else:
        draw_matrix(a, m)
#print_m(m)
#print(the_path)
saveas = './outputs/' + output_name +'.gif'
images[0].save(saveas,
               save_all=True, append_images=images[1:],
               optimize=False, duration=1, loop=0)
print("Output generated. Close program and check working directory.")
```

TIME COMPLEXITY ANALYSIS

The time complexity of the given code that converts an image of a maze into a 2D array and solves it using breadth-first search can be analyzed as follows:

1. Reading and manipulating the maze image file using the PIL library takes O(n) time, where n is the number of pixels in the image.

2. Converting the image into a 2D array takes O(n) time, as each pixel needs to be processed.

3. The breadth-first search algorithm used to solve the maze has a time complexity of

O (V + E), where V is the number of vertices (or cells) in the maze, and E is the number of edges (or paths) between those vertices. In the worst case, where the maze is a perfect grid and every cell is connected to its four neighbors, V = n and E = 2n - 2, giving a time complexity of O(n).

4. Drawing the output image using the PIL ImageDraw library takes O(n) time, as each pixel needs to be processed.

Therefore, the overall time complexity of the code can be approximated as O(n) for large mazes.

RESULT

The code is a Python program that converts an image of a maze into a 2D array and solves it using breadth-first search. It uses the PIL library to manipulate the image, and the output is visualized using the PIL ImageDraw library. The program prompts the user to input the maze image file name and output file name. The resulting image shows the solved maze with a red line representing the path from the start point to the end point. Therefore, the maze is solved using Breadth First Search Algorithm.

REFERENCES

[1] K. Meijer, "Maze Generation," [Online]. Available: https://keesiemeijer.github.io/maze-generator/#generate.

[2] S. Adhikari, "Solve a Maze with Python," Level Up Coding, 2019. [Online]. Available: https://levelup.gitconnected.com/solve-a-maze-with-python-e9f0580979a1.