Experiment-2

Yash Patel, Ojas Patil
2019130047 , 2019130048
TE Comps
Batch C

Aim: To implement the fire extinguisher using BFS and DFS

Theory:

- Depth-first Search (DFS):
    1. DFS always expands DEPTH-FIRST the deepest node in the current frontier of the search tree.
    2. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
    3. DFS uses a LIFO queue.
    4. Visits children before siblings.

- Breadth-first Search (BFS):
    1. BFS is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
    2. All the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
    3. BFS uses a FIFO queue.
    4. Visits siblings before children.

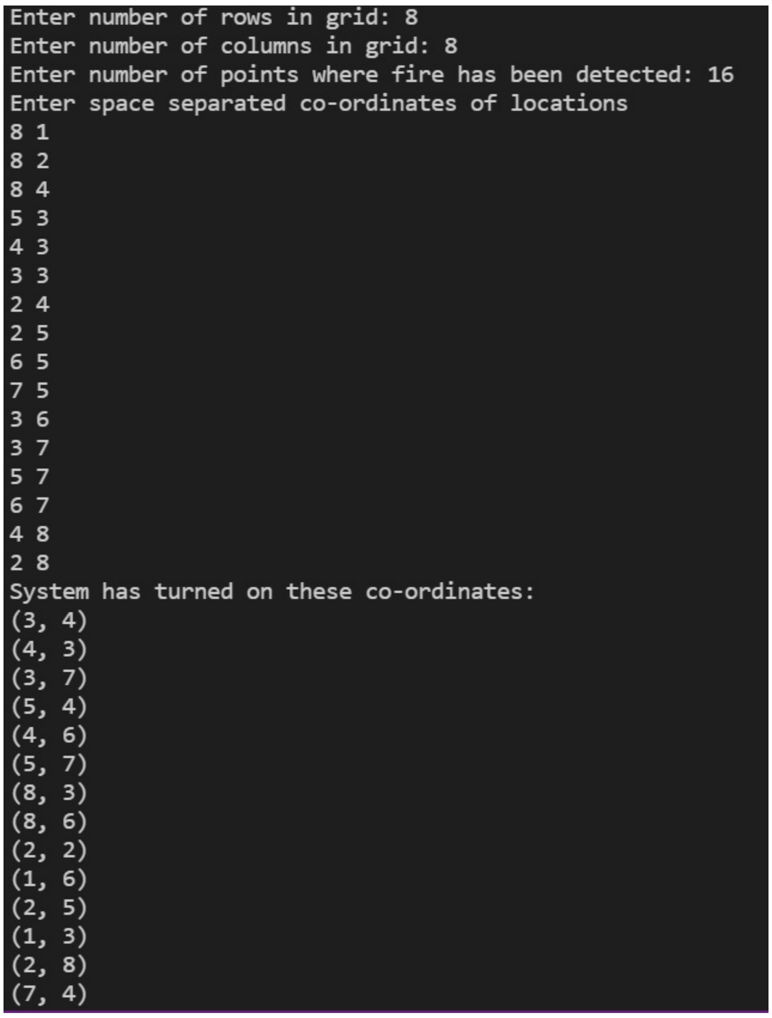
Code:

```
# tree.py
class Node:
  def __init__(self):
    self.fire = False # If fire is there in this node, turn self.fire = True
    self.DFS = False # If you explore this node using BFS, turn self.BFS = True
    self.BFS = False # If you explore this node using DFS, turn self.DFS = True
    self.dis = -1

class Holder:
  def __init__(self, x, y) -> None:
    self.x = x
    self.y = y

# fireExtinguisherRescuer.py
import copy
from tree import *

dx = [0, 1, 0, -1]
dy = [-1, 0, 1, 0]

grid_size_length = int(input("Enter number of rows in grid: "))
grid_size_breadth = int(input("Enter number of columns in grid: "))
```

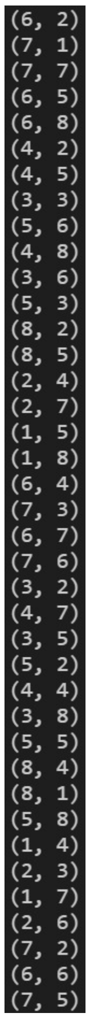
```
grid = [[None for _ in range(grid_size_breadth)] for _ in range(grid_size_length)]

for i in range(grid_size_length):
  for j in range(grid_size_breadth):
    grid[i][j] = Node()

no_of_points = int(input("Enter number of points where fire has been detected: "))
print("Enter space separated co-ordinates of locations")

for i in range(no_of_points):
  point = tuple(map(int, input().split()))
  grid[point[0] - 1][point[1] - 1].fire = True

def printer():
  for i in grid:
    print(i)

def checkBound(x, y):
  if x >= 0 and y >= 0 and x < grid_size_length and y < grid_size_breadth and (grid[x][y]).fire ==
False:
    return True
  else:
    return False

# doing BFS
def BFS():
  visited = [[False for _ in range(grid_size_length)] for _ in range(grid_size_breadth)]

  queue = []
  extinguishers_turned_on = []
  no_of_extinguishers = 0
  # FIFO -> first in first out

  for i in range(grid_size_length):
    for j in range(grid_size_breadth):
      if grid[i][j].fire == True:
        no_of_extinguishers += 1
        queue.append((i, j))

  dx = [0, 0, -1, 1, 1, -1, 1, -1]
  dy = [-1, 1, 0, 0, 1, 1, -1, -1]

  while len(queue) > 0:
    current = queue[0]
    queue.pop(0)
    x = current[0]
    y = current[1]
    visited[x][y] = 1
    for i in range(8):
      xx = x + dx[i]
      yy = y + dy[i]
      if xx < 0 or yy < 0 or xx >= grid_size_length or yy >= grid_size_breadth or grid[xx][yy] ==
False or visited[xx][yy]:
        continue
      extinguishers_turned_on.append((xx + 1, yy + 1))
      # no_of_extinguishers += 1
  return extinguishers_turned_on

def _BFS(x, y):
  found = False
  queue = []
  cur = Holder(x, y)
  (grid[cur.x][cur.y]).dis = 0
  queue.append(cur)
  while len(queue) > 0:
    cur = queue[0]
    queue.pop(0)
    if (cur.x == grid_size_length - 1) and (cur.y == grid_size_breadth - 1):
      found = True
      # DFS()
    for i in range(4):
      nt = copy.deepcopy(cur)
      nt.x += dx[i]
      nt.y += dy[i]
      if checkBound(nt.x, nt.y) and grid[nt.x][nt.y].dis == -1:
        (grid[nt.x][nt.y]).dis = (grid[cur.x][cur.y]).dis + 1
        queue.append(nt)

  return found

path = []
ans = []
def DFS(stuck_co_ordinate, x, y, d):
  if x == stuck_co_ordinate[0] and y == stuck_co_ordinate[1]:
    onePath = list()
    onePath.append(tuple((4, 5)))
    for i in range(d-1, -1, -1):
      onePath.append(ans[i])

    path.append(onePath)

    return

  for i in range(4):
    nx = x + dx[i]
    ny = y + dy[i]
    if checkBound(nx, ny) and (grid[x][y]).dis - 1 == (grid[nx][ny]).dis:
      ans.append(tuple((nx, ny)))
      DFS(stuck_co_ordinate, nx, ny, d+1)
      ans.pop()
```

```
extinguishers_turned_on = BFS()
print("System has turned on these co-ordinates: ")
for co_ordinate in extinguishers_turned_on:
  print(co_ordinate)

stuck_co_ordinate = tuple(map(int, input("Are you stuck? tell us your location: ").strip().split()))
stuck_co_ordinate = tuple((stuck_co_ordinate[0] - 1, stuck_co_ordinate[1] - 1))
if not _BFS(stuck_co_ordinate[0], stuck_co_ordinate[1]):
  print("""Sorry to inform you that entrance has caught the fire!!\n
  Please stay there until our fire extinguishers extinguish the fire.\n
  It won't take long and keep yourself away from flames.""")
else:
  DFS(stuck_co_ordinate, grid_size_length - 1, grid_size_breadth - 1, 0)
  if len(path) > 0:
    print("Follow one of these paths")
    for i in path:
      i.append(i.pop(0))
      print(i)
```

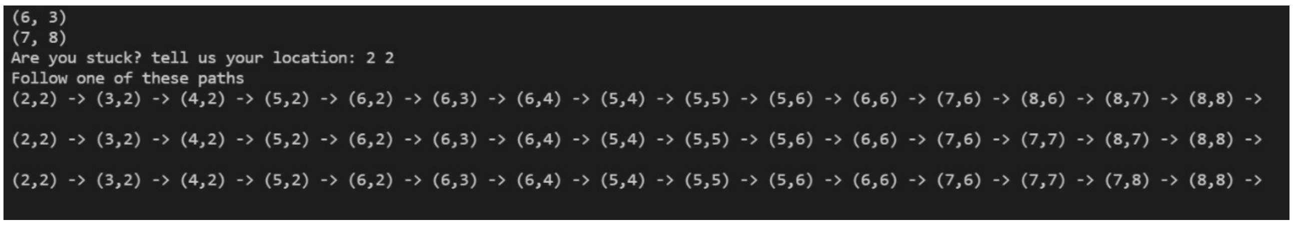
Input/Output:



```
Enter number of rows in grid: 8
Enter number of columns in grid: 8
Enter number of points where fire has been detected: 16
Enter space separated co-ordinates of locations
8 1
8 2
8 4
5 3
4 3
3 3
2 4
2 5
6 5
7 5
3 6
3 7
5 7
6 7
4 8
2 8
System has turned on these co-ordinates:
(3, 4)
(4, 3)
(3, 7)
(5, 4)
(4, 6)
(5, 7)
(8, 3)
(8, 6)
(2, 2)
(1, 6)
(2, 5)
(1, 3)
(2, 8)
(7, 4)
```



```
(6, 2)
(7, 1)
(7, 7)
(6, 5)
(6, 8)
(4, 2)
(4, 5)
(3, 3)
(5, 6)
(4, 8)
(3, 6)
(5, 3)
(8, 2)
(8, 5)
(2, 4)
(2, 7)
(1, 5)
(1, 8)
(6, 4)
(7, 3)
(6, 7)
(7, 6)
(3, 2)
(4, 7)
(3, 5)
(5, 2)
(4, 4)
(3, 8)
(5, 5)
(8, 4)
(8, 1)
(5, 8)
(1, 4)
(2, 3)
(1, 7)
(2, 6)
(7, 2)
(6, 6)
(7, 5)
```



```
(6, 3)
(7, 8)
Are you stuck? tell us your location: 2 2
Follow one of these paths
(2,2) -> (3,2) -> (4,2) -> (5,2) -> (6,2) -> (6,3) -> (6,4) -> (5,4) -> (5,5) -> (5,6) -> (6,6) -> (7,6) -> (8,6) -> (8,7) -> (8,8) ->

(2,2) -> (3,2) -> (4,2) -> (5,2) -> (6,2) -> (6,3) -> (6,4) -> (5,4) -> (5,5) -> (5,6) -> (6,6) -> (7,6) -> (7,7) -> (8,7) -> (8,8) ->

(2,2) -> (3,2) -> (4,2) -> (5,2) -> (6,2) -> (6,3) -> (6,4) -> (5,4) -> (5,5) -> (5,6) -> (6,6) -> (7,6) -> (7,7) -> (7,8) -> (8,8) ->
```

Conclusion:
The environment for this experiment is a floor, and it is divided into a grid of size NxM. Each node has a pair of extinguishers and a sensor. If any of the nodes catches fire, it begins the BFS search to find all the nodes in its very next frontier and activates the fire extinguishers so that the fire can be contained and extinguished.If given the location of a person who has been stuck in any of the nodes, from that location another BFS search is executed to find the length shortest path to the exit door located at the lower right corner of the room. Since there could be multiple shortest paths, we provide the person with all the possible paths using DFS, so that person can rescue himself/herself from the room. Hence, we learnt to implement DFS and BFS algorithms.

Links:
- Yash Patel: https://github.com/yash19pro/AI-ML-Lab
- Ojas Patil: https://github.com/PatilOjas/AIML-Lab