ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

BackTracking

Lớp: CS112.O23

GVHD: Nguyễn Thanh Sơn

Sinh viên: Nguyễn Trần Khương An

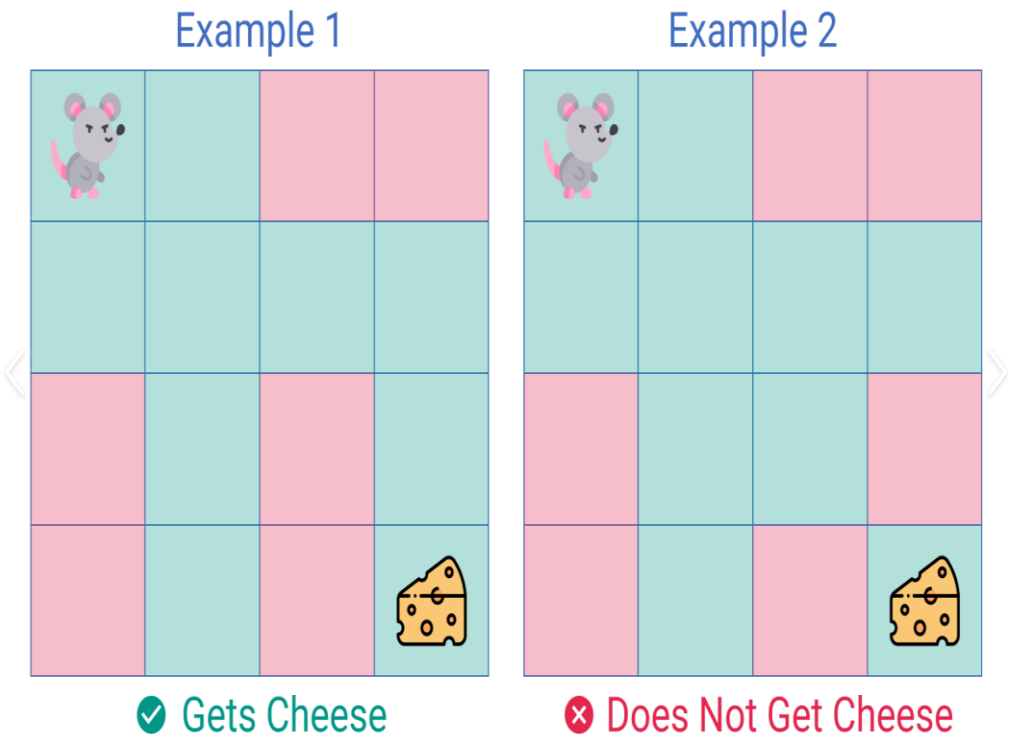
MSSV: 22520026

TP Hồ Chí Minh, Tháng 4 năm 2024

I. Bài toán

1. Yêu cầu

Xây dựng thuật toán tìm đường đi trong mê cung cho bài toán con chuột tìm đến miếng pho mát – Rat in a Maze.



2. Bài toán

Xét một con chuột được đặt ở vị trí **(0, 0)** trong ma trận vuông cấp **N * N** . Nó phải đến đích tại **(N – 1, N – 1)** . Tìm tất cả các con đường có thể mà chuột có thể đi từ nguồn đến đích. Các hướng mà chuột có thể di chuyển là **'U'(Up)**, **'D'(Down)** , **'L' (Left)**, **'R' (Right)**. Giá trị **0** tại một ô trong ma trận biểu thị rằng nó bị chặn và chuột không thể di chuyển đến ô đó trong khi giá trị **1** tại một ô trong ma trận biểu thị rằng chuột có thể di chuyển qua ô đó. Hãy in ra tất cả các đường đi có thể để giúp con chuột đi đến vị trí miến pho mát.

Lưu ý : Trong một đường đi, không ô nào có thể được truy cập nhiều lần. Nếu ô nguồn bằng **0** thì chuột không thể di chuyển sang bất kỳ ô nào khác.

Input:

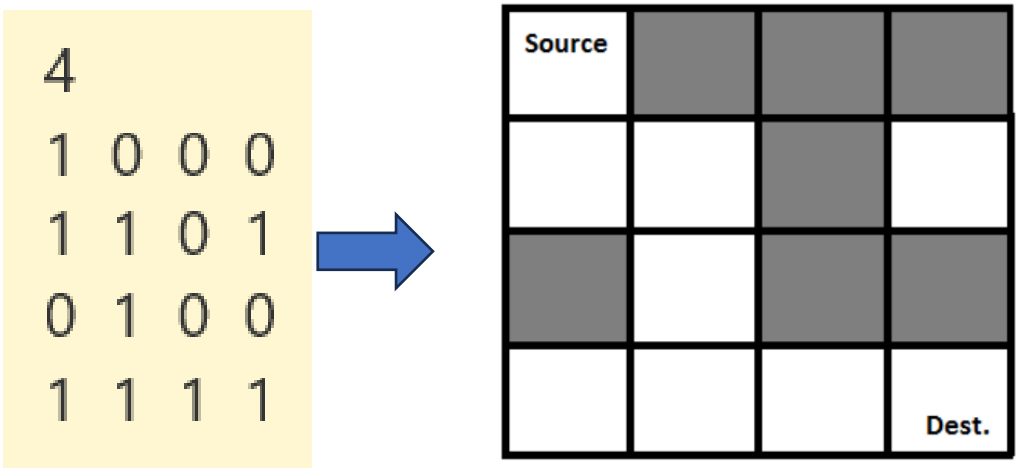
- Số nguyên N để biểu diễn mê cung NxN.
- N dòng tiếp theo mỗi dòng chứa N phần tử gồm 2 giá trị 0 và 1. 0 đại diện cho ô bị chặn và 1 đại diện cho ô trống.

Output:

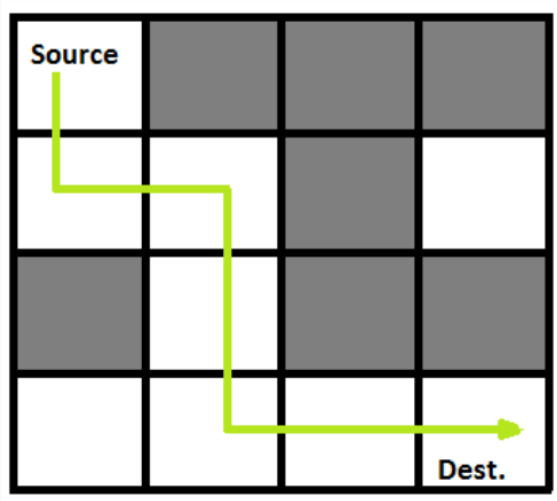
- In ra tất cả đường đi cho con chuột để nó có thể di chuyển từ vị trí bắt đầu đến vị trí miếng pho mát.

Ví dụ:

- **Input**



- **Output:** DRDDRR



II. Ý tưởng

- Sử dụng thuật toán quay lui - BackTracking để khám phá tất cả các đường đi có thể.
- Các bước triển khai:
  - Tạo hàm **isValid()** để kiểm tra xem một ô ở vị trí (hàng, cột) có nằm trong mê cung và được bỏ chặn hay không.
  - Tạo **findPath()** để nhận tất cả các đường đi hợp lệ:
    - **Trường hợp cơ sở:** Nếu vị trí hiện tại là ô dưới cùng bên phải thì thêm đường đi hiện tại vào kết quả và trả về.
    - Đánh dấu ô hiện tại là đã thăm.

- Lặp lại thông qua tất cả các hướng có thể.
    - Tính toán vị trí tiếp theo dựa trên hướng hiện tại.
    - Nếu vị trí tiếp theo hợp lệ (tức là nếu **isValid()** trả về **true**), hãy thêm hướng vào đường dẫn hiện tại và gọi đệ quy hàm **findPath()** cho ô tiếp theo.
    - Quay lại bằng cách xóa hướng cuối cùng khỏi đường dẫn hiện tại.
- Đánh dấu ô hiện tại là đã bỏ chặn trước khi quay lại.

III. Cài đặt

1. Khởi tạo hướng và mảng thay đổi vị trí

```
# Initialize a string direction which represents all the directions.
direction = "DLRU"

# Arrays to represent change in rows and columns
dr = [1, 0, 0, -1]
dc = [0, -1, 1, 0]
```

2. Hàm kiểm tra vị trí hợp lệ

```
# Function to check if cell(row, col) is inside the maze and unblocked
def is_valid(row, col, n, maze):
    return 0 <= row < n and 0 <= col < n and maze[row][col] == 1
```

3. Hàm quay lui tìm kiếm tất cả đường đi

```
# Function to get all valid paths
def find_path(row, col, maze, n, ans, current_path):
    # If we reach the bottom right cell of the matrix, add
    # the current path to ans and return
    if row == n - 1 and col == n - 1:
        ans.append(current_path)
        return
    # Mark the current cell as blocked
    maze[row][col] = 0

    for i in range(4):
        # Find the next row based on the current row (row) and the dr[] array
        next_row = row + dr[i]
        # Find the next column based on the current column (col) and the dc[] array
        next_col = col + dc[i]

        # Check if the next cell is valid or not
        if is_valid(next_row, next_col, n, maze):
            current_path += direction[i]
            # Recursively call the find_path function for the next cell
            find_path(next_row, next_col, maze, n, ans, current_path)
            # Remove the last direction when backtracking
            current_path = current_path[:-1]

    # Mark the current cell as unblocked
    maze[row][col] = 1
```

4. Hàm Main

```
# Driver code
maze = [
    [1, 0, 0, 0],
    [1, 1, 0, 1],
    [1, 1, 0, 0],
    [0, 1, 1, 1]
]

n = len(maze)
# List to store all the valid paths
result = []
# Store current path
current_path = ""

if maze[0][0] != 0 and maze[n - 1][n - 1] != 0:
    # Function call to get all valid paths
    find_path(0, 0, maze, n, result, current_path)

if not result:
    print(-1)
else:
    print(" ".join(result))
```