TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA Công Nghệ Thông Tin
BỘ MÔN: Công Nghệ Phần Mềm

ĐỀ THI VÀ BÀI LÀM

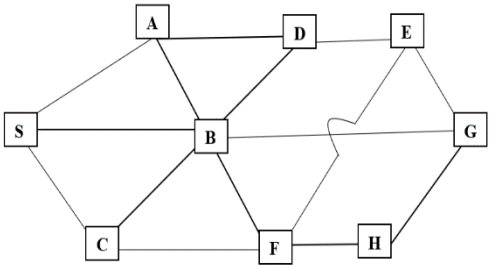
Tên học phần: Trí tuệ nhân tạo
Mã học phần: Hình thức thi: *Tự luận có giám sát*
Đề số: **01** Thời gian làm bài: 75 phút *(không kể thời gian chép/phát đề)*
Được sử dụng tài liệu khi làm bài.

Họ tên: Nguyễn Hữu Khoa……**Lớp**:…22T_DT5……**MSSV**:…102220237

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV_HọTên.pdf và nộp bài thông qua MSTeam:

Lưu ý: Mã sinh viên nếu số chẵn thì làm BFS ở câu 1), còn mã ssinh viên lẽ thì làn DFS ở câu 1).

Câu 1 (5 *điểm*): Cho đồ thị vô hướng G = (V,E) như hình vẽ với V là tập đỉnh và E là tập cạnh.



a) *(1 điểm)* Hãy viết đoạn code biểu diễn đồ thị trên bằng cách khởi tạo tập đỉnh V và tập cạnh E.

 # Trả lời: Dán code vào bên dưới

```
import matplotlib.pyplot as plt
import networkx as nx
from collections import deque

# Create a graph object
G = nx.Graph()

# Add V
#tập đỉnh
V = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'S']
G.add_nodes_from(V)

# tập cạnh
# Add edges
E = [
    ('A', 'B'),
    ('A', 'D'),
    ('A', 'S'),
```

```
    ('B', 'C'),
    ('B', 'F'),
    ('B', 'G'),
    ('B', 'S'),
    ('B', 'D'),

    ('C', 'S'),
    ('C', 'F'),

    ('D', 'E'),

    ('E', 'G'),
    ('E', 'F'),

    ('F', 'H'),
    ('G', 'H')
]
G.add_edges_from(E)

# Draw the graph
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(G, seed=10)  # Adjust layout as needed
nx.draw_networkx_nodes(G, pos, node_size=500, node_color='lightblue')
nx.draw_networkx_edges(G, pos, width=2)
nx.draw_networkx_labels(G, pos, font_size=12, font_family='sans-serif')

# Highlight the path
# if path:
#     edge_path = list(zip(path, path[1:]))
#     nx.draw_networkx_edges(G, pos, edgelist=edge_path, width=2, edge_color='r')

plt.title('Graph Visualization')
plt.axis('off')
plt.show()
```

b) *(3 điểm)* Hãy viết chương trình sử dụng thuật toán **BFS/DFS** để tìm đường đi từ. Trong chương trình, hãy in ra thứ tự đỉnh khám phá trong quá trình tìm kiếm. Nếu không tìm thấy thì in "*Khong tim thay duong di*"

 # Trả lời: Dán code vào bên dưới

```
import matplotlib.pyplot as plt
import networkx as nx
from collections import deque

# Create a graph object
G = nx.Graph()

# Add V
#tập đỉnh
V = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'S']
G.add_nodes_from(V)

# tập cạnh
# Add edges
E = [
    ('A', 'B'),
```

```
    ('A', 'D'),
    ('A', 'S'),

    ('B', 'C'),
    ('B', 'F'),
    ('B', 'G'),
    ('B', 'S'),
    ('B', 'D'),

    ('C', 'S'),
    ('C', 'F'),

    ('D', 'E'),

    ('E', 'G'),
    ('E', 'F'),

    ('F', 'H'),
    ('G', 'H')
]
G.add_edges_from(E)

def dfs(graph, start, goal):
    stack = [(start, [start])]
    visited = set()
    all_paths = []

    while stack:
        (vertex, path) = stack.pop()
        if vertex in visited:
            continue

        visited.add(vertex)
        all_paths.append(path)

        for neighbor in graph.neighbors(vertex):
            if neighbor == goal:
                all_paths.append(path + [neighbor])
                print("Visited paths:", all_paths)
                return path + [neighbor]
            else:
                stack.append((neighbor, path + [neighbor]))
    if(all_paths):
        print("Visited paths:", all_paths)
    else:
        print("Khong tim thay duong di")
    return None

# Find path from 'S' to 'G'
path = dfs(G, 'G', 'S')
print("Path from G to S:", path)

# Draw the graph
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(G, seed=10)  # Adjust layout as needed
nx.draw_networkx_nodes(G, pos, node_size=500, node_color='lightblue')
nx.draw_networkx_edges(G, pos, width=2)
nx.draw_networkx_labels(G, pos, font_size=12, font_family='sans-serif')

# Highlight the path
# if path:
#     edge_path = list(zip(path, path[1:]))
#     nx.draw_networkx_edges(G, pos, edgelist=edge_path, width=2, edge_color='r')
```

```
plt.title('Graph Visualization')
plt.axis('off')
plt.show()
```

 # Trả lời: Dán kết quả thực thi vào bên dưới:

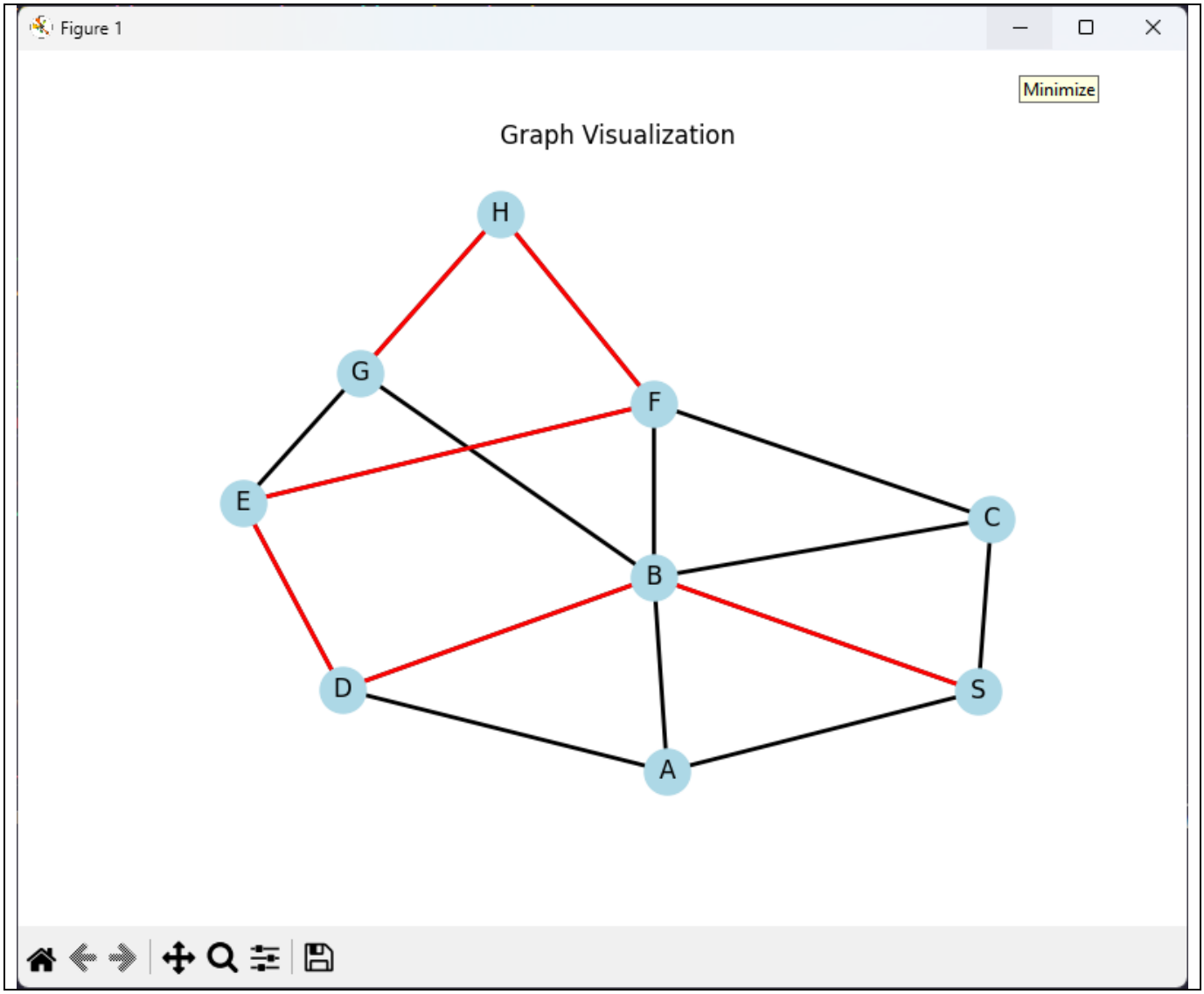
```
Visited paths: [['G'], ['G', 'H'], ['G', 'H', 'F'], ['G', 'H', 'F', 'E'], ['G', 'H', 'F', 'E', 'D'], ['G', 'H', 'F', 'E', 'D',
'B'], ['G', 'H', 'F', 'E', 'D', 'B', 'S']]
Path from G to S: ['G', 'H', 'F', 'E', 'D', 'B', 'S']
```

c) *(1 điểm)* Hãy trực quan hóa kết quả tìm kiếm đường đi (nếu BFS thì tìm từ đỉnh "S" đến đỉnh "G", còn nếu DFS thì tìm từ G tới S)

 # Trả lời: Dán code vào bên dưới

```
import matplotlib.pyplot as plt
import networkx as nx
from collections import deque

# Create a graph object
G = nx.Graph()

# Add V
#tập đỉnh
V = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'S']
G.add_nodes_from(V)

# tập cạnh
# Add edges
E = [
    ('A', 'B'),
    ('A', 'D'),
    ('A', 'S'),

    ('B', 'C'),
    ('B', 'F'),
    ('B', 'G'),
    ('B', 'S'),
    ('B', 'D'),

    ('C', 'S'),
    ('C', 'F'),

    ('D', 'E'),

    ('E', 'G'),
    ('E', 'F'),

    ('F', 'H'),
    ('G', 'H')
]
G.add_edges_from(E)

def dfs(graph, start, goal):
    stack = [(start, [start])]
    visited = set()
    all_paths = []
```

```
    while stack:
        (vertex, path) = stack.pop()
        if vertex in visited:
            continue

        visited.add(vertex)
        all_paths.append(path)

        for neighbor in graph.neighbors(vertex):
            if neighbor == goal:
                all_paths.append(path + [neighbor])
                print("Visited paths:", all_paths)
                return path + [neighbor]
            else:
                stack.append((neighbor, path + [neighbor]))

    print("Visited paths:", all_paths)
    return None

# Find path from 'S' to 'G'
path = dfs(G, 'G', 'S')
print("Path from G to S:", path)

# Draw the graph
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(G, seed=10)  # Adjust layout as needed
nx.draw_networkx_nodes(G, pos, node_size=500, node_color='lightblue')
nx.draw_networkx_edges(G, pos, width=2)
nx.draw_networkx_labels(G, pos, font_size=12, font_family='sans-serif')

# Highlight the path
if path:
    edge_path = list(zip(path, path[1:]))
    nx.draw_networkx_edges(G, pos, edgelist=edge_path, width=2, edge_color='r')

plt.title('Graph Visualization')
plt.axis('off')
plt.show()
```
 # Trả lời: Dán hình ảnh trực quan hoá vào bên dưới



Câu 2 (*5 điểm*): Trong các lâu đài cổ người ta thường xây dựng các đường hầm bí mật để thoát hiểm trong các trường hợp khẩn cấp. Các đường hầm chỉ có thể vào từ một cửa vào duy nhất tại phòng Trung tâm và thoát ra ở rất nhiều cửa ra. Các cửa ra đều nằm ở rìa lâu đài, do vậy, nếu thoát ra được rìa lâu đài thì coi như đã thoát hiểm. Để ngụy trang, người ta cho đào nhiều nhánh hầm cụt và cửa vào giả. Ngoài ra, để tăng khả năng thoát hiểm, người ta còn xây dựng các đường hầm giao nhau tại một số vị trí. Để nghiệm thu công trình, chủ lâu đài cần kiểm tra xem từ phòng trung tâm có thể thoát hiểm qua hệ thống đường hầm hay không.

Cho biết dữ liệu lâu đài là một hình vuông được chia lưới ô vuông gồm n dòng, n cột. Trên đồ hoạ, ô ở dòng i cột j được ghi số 1 nếu có đường hầm, số 0 nếu không có (ô ở góc trên trái có toạ độ (0,0)). 2 ô chỉ có thể thông nhau nếu chúng có chung cạnh.

Minh hoạ dữ liệu tập tin (.csv) gồm

4 2 1

0 1 1 0

1 0 0 1

1 1 1 1

0 1 1 0

- Dòng đầu chứa 3 số nguyên dương n, D và C (trong đó D, C là dòng và cột của phòng trung tâm).

- n dòng tiếp theo, mỗi dòng chứa n số là các số ở các vị trí tương ứng trên họa đồ.

a) (2 điểm) Hãy trình bày thuật toán A* bằng sơ đồ khối (khuyến khích vẻ bằng io draw)

 # Trả lời: Dán sơ đồ khối vào bên dưới

b) (2 điểm)Hãy viết hàm sử dụng A* để giúp chủ lâu đài kiểm tra hệ thống trên.

 # Trả lời: Dán hàm vào bên dưới

```
import math
import heapq
import csv


class Cell:
    def __init__(self):
        self.parent_i = 0
        self.parent_j = 0
        self.f = float('inf')
        self.g = float('inf')
        self.h = 0


ROW = 5
COL = 4


def is_valid(row, col):
    return 0 <= row < ROW and 0 <= col < COL


def is_unblocked(grid, row, col):
    return grid[row][col] == 1


def is_destination(row, col, dest):
    return row == dest[0] and col == dest[1]


def calculate_h_value(row, col, dest):
```

```
    return math.sqrt((row - dest[0]) ** 2 + (col - dest[1]) ** 2)


def trace_path(cell_details, dest):
    print("The Path is ")
    path = []
    row, col = dest

    while not (cell_details[row][col].parent_i == row and cell_details[row][col].parent_j == col):
        path.append((row, col))
        row, col = cell_details[row][col].parent_i, cell_details[row][col].parent_j

    path.append((row, col))
    path.reverse()

    for i in path:
        print("->", i, end=" ")
    print()

def a_star_search(grid, src, dest):
    if not is_valid(src[0], src[1]) or not is_valid(dest[0], dest[1]):
        print("Source or destination is invalid")
        return

    if not is_unblocked(grid, src[0], src[1]) or not is_unblocked(grid, dest[0], dest[1]):
        print("Source or the destination is blocked")
        return

    if is_destination(src[0], src[1], dest):
        print("We are already at the destination")
        return

    closed_list = [[False for _ in range(COL)] for _ in range(ROW)]
```

```
    cell_details = [[Cell() for _ in range(COL)] for _ in range(ROW)]

    i, j = src
    cell_details[i][j].f = cell_details[i][j].g = cell_details[i][j].h = 0
    cell_details[i][j].parent_i = cell_details[i][j].parent_j = i

    open_list = []
    heapq.heappush(open_list, (0.0, i, j))

    found_dest = False

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)]

    while open_list:
        p = heapq.heappop(open_list)
        i, j = p[1], p[2]
        closed_list[i][j] = True

        for dir in directions:
            new_i, new_j = i + dir[0], j + dir[1]

            if is_valid(new_i, new_j) and is_unblocked(grid, new_i, new_j) and not
closed_list[new_i][new_j]:
                if is_destination(new_i, new_j, dest):
                    cell_details[new_i][new_j].parent_i = i
                    cell_details[new_i][new_j].parent_j = j
                    print("The destination cell is found")
                    trace_path(cell_details, dest)
                    found_dest = True
                    return
                else:
                    g_new = cell_details[i][j].g + 1.0
```

```
                h_new = calculate_h_value(new_i, new_j, dest)
                f_new = g_new + h_new

                if cell_details[new_i][new_j].f == float('inf') or cell_details[new_i][new_j].f > f_new:
                    heapq.heappush(open_list, (f_new, new_i, new_j))
                    cell_details[new_i][new_j].f = f_new
                    cell_details[new_i][new_j].g = g_new
                    cell_details[new_i][new_j].h = h_new
                    cell_details[new_i][new_j].parent_i = i
                    cell_details[new_i][new_j].parent_j = j

    if not found_dest:
        print("Failed to find the destination cell")

def main():
    grid = []
    with open('data.csv', 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            grid.append([int(cell) for cell in row])

    src = [4, 0]
    dest = [0, 0]

    a_star_search(grid, src, dest)

if __name__ == "__main__":
    main()
```

c) (1 điểm)Viết chương trình hoàn thiện cho bài toán trên và Hãy in kết quả tìm được trên màn hình. Dòng đầu chứa số m là số ô phải đi qua, nếu không thoát được thì m = -1. Trong trường hợp thoát được, m dòng tiếp theo: mỗi dòng chứa 2 số là số hiệu dòng cột của các ô phải đi qua theo đúng trình tự của một cách thoát hiểm.

 # Trả lời: Dán code vào bên dưới

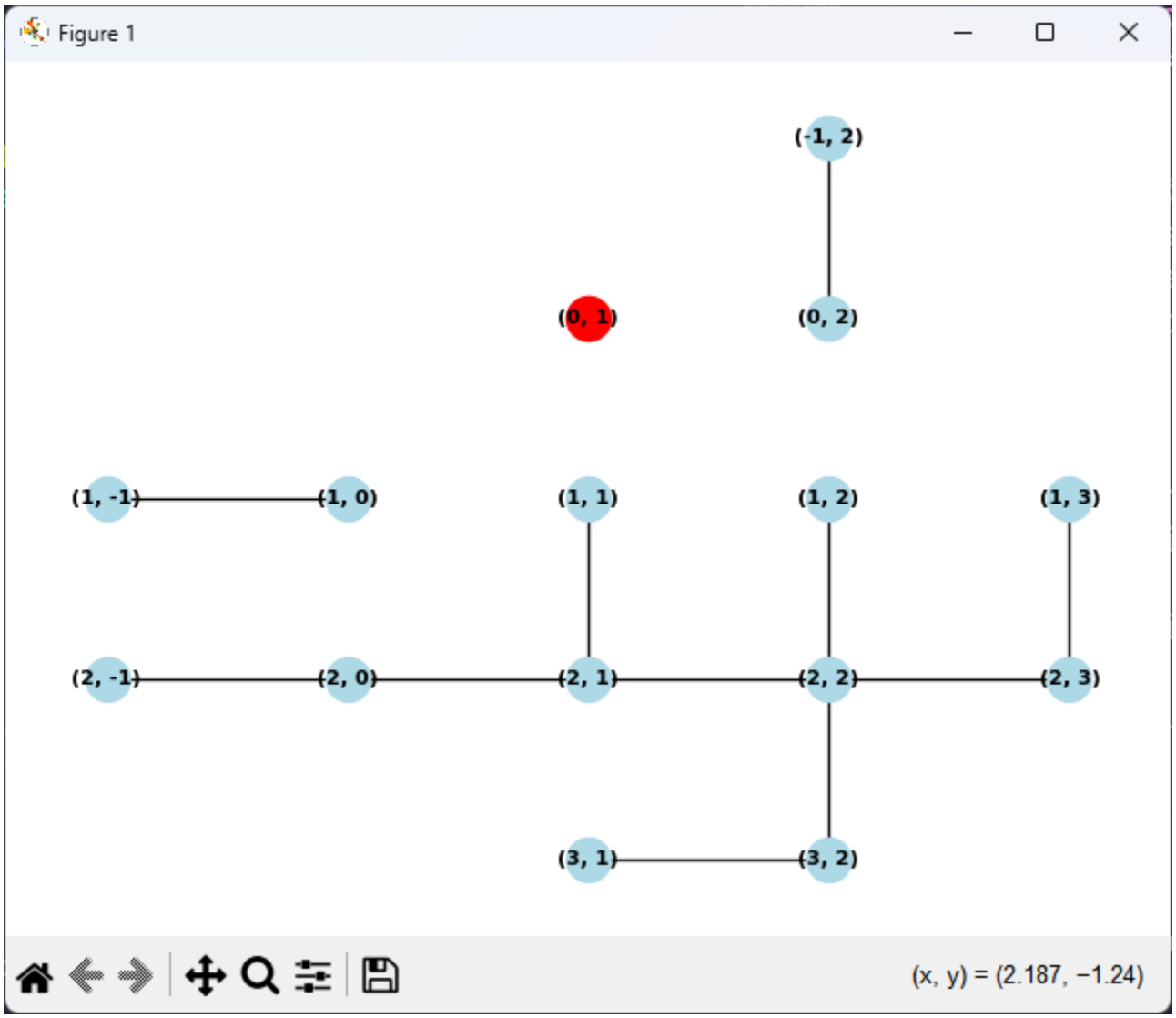
```
import networkx as nx
import csv
import heapq
import matplotlib.pyplot as plt


def read_data(filename):
    """Đọc dữ liệu từ file CSV và tạo đồ thị"""
    G = nx.Graph()
    with open(filename, 'r') as f:
        reader = csv.reader(f)
        rows, cols, start_row = next(reader)
        rows, cols, start_row = int(rows), int(cols), int(start_row) - 1
        start_col = int(cols / 2)


        for i, row in enumerate(reader):
            for j, val in enumerate(row):
                if val == '1':
                    G.add_node((i, j))
                    if i > 0 and row[j-1] == '1':
                        G.add_edge((i, j), (i, j-1))
                    if j > 0 and row[j-1] == '1':
                        G.add_edge((i, j), (i-1, j))


        return G, (start_row, start_col)


def heuristic(a, b):
    """Hàm heuristic tính khoảng cách Manhattan giữa hai điểm a và b"""
    return abs(a[0] - b[0]) + abs(a[1] - b[1])


def a_star_search(graph, start, goal):
    """Thuật toán A* tìm đường đi ngắn nhất từ start đến goal"""
```

```
    open_set = []
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {node: float('inf') for node in graph.nodes}
    g_score[start] = 0
    f_score = {node: float('inf') for node in graph.nodes}
    f_score[start] = heuristic(start, goal)

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            return reconstruct_path(came_from, current)

        for neighbor in graph.neighbors(current):
            tentative_g_score = g_score[current] + 1  # Giả sử mỗi cạnh có trọng số bằng 1

            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic(neighbor, goal)
                if neighbor not in [i[1] for i in open_set]:
                    heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None

def reconstruct_path(came_from, current):
    """Hàm tái tạo đường đi từ start đến goal"""
    total_path = [current]
    while current in came_from:
        current = came_from[current]
        total_path.append(current)
```

```
    return total_path[::-1]


def draw_graph(G, path=None):
    """Vẽ đồ thị sử dụng NetworkX và Matplotlib"""
    pos = {node: (node[1], -node[0]) for node in G.nodes}  # Đảo ngược trục y để vẽ đúng hướng
    nx.draw(G, pos, with_labels=True, node_size=300, node_color='lightblue', font_size=8,
font_weight='bold')


    if path:
        path_edges = list(zip(path, path[1:]))
        nx.draw_networkx_nodes(G, pos, nodelist=path, node_color='red')
        nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='red', width=2)


    plt.show()


def main():
    G, start = read_data('data.csv')


    # Tìm tất cả các ô biên
    rows = max(node[0] for node in G.nodes) + 1
    cols = max(node[1] for node in G.nodes) + 1
    goals = [(i, 0) for i in range(rows)] + [(i, cols-1) for i in range(rows)] + \
            [(0, j) for j in range(1, cols-1)] + [(rows-1, j) for j in range(1, cols-1)]


    # Tìm đường đi ngắn nhất đến bất kỳ ô biên nào
    shortest_path = None
    for goal in goals:
        path = a_star_search(G, start, goal)
        if path and (not shortest_path or len(path) < len(shortest_path)):
            shortest_path = path


    if shortest_path is None:
```

```
        print(-1)
    else:
        print(len(shortest_path) - 1)  # Số bước đi
        for step in shortest_path:
            print(step)

    # Vẽ đồ thị và đường đi ngắn nhất
    draw_graph(G, shortest_path)

if __name__ == "__main__":
    main()
```

 # Trả lời: Dán hình ảnh kết quả vào bên dưới(với dữ liệu minh hoạ ở trên)



Đà Nẵng, ngày 10 tháng 10 năm 2024

GIẢNG VIÊN BIÊN SOẠN ĐỀ THI

TRƯỞNG BỘ MÔN
(đã duyệt)