Дискретні структури

ІПЗ-21-2
Губарєв Ростислав Вадимович

Лабораторна робота №12.1

Вихід із лабіринту.

Поле лабіринту було розбито на клітини, після чого в прямокутну матрицю NxM було занесено інформацію про кожну клітину: 0, якщо переміщення через клітину можливе, і 1, якщо ні.
Значенням "10" позначено вихід із лабіринту. На цій карті задано початкове положення гравця у вигляді координат клітини, де він перебуває. Гравець може переміщатися в сусідні клітини вгору, вниз, вліво або вправо.

Уявіть лабіринт у вигляді графа і візуалізуйте його. Знайдіть найкоротший шлях гравця до виходу з лабіринту і намалюйте цей шлях на графі.
Якщо таких шляхів кілька - виведіть будь-який із них. Якщо шляху немає - виведіть повідомлення про це.

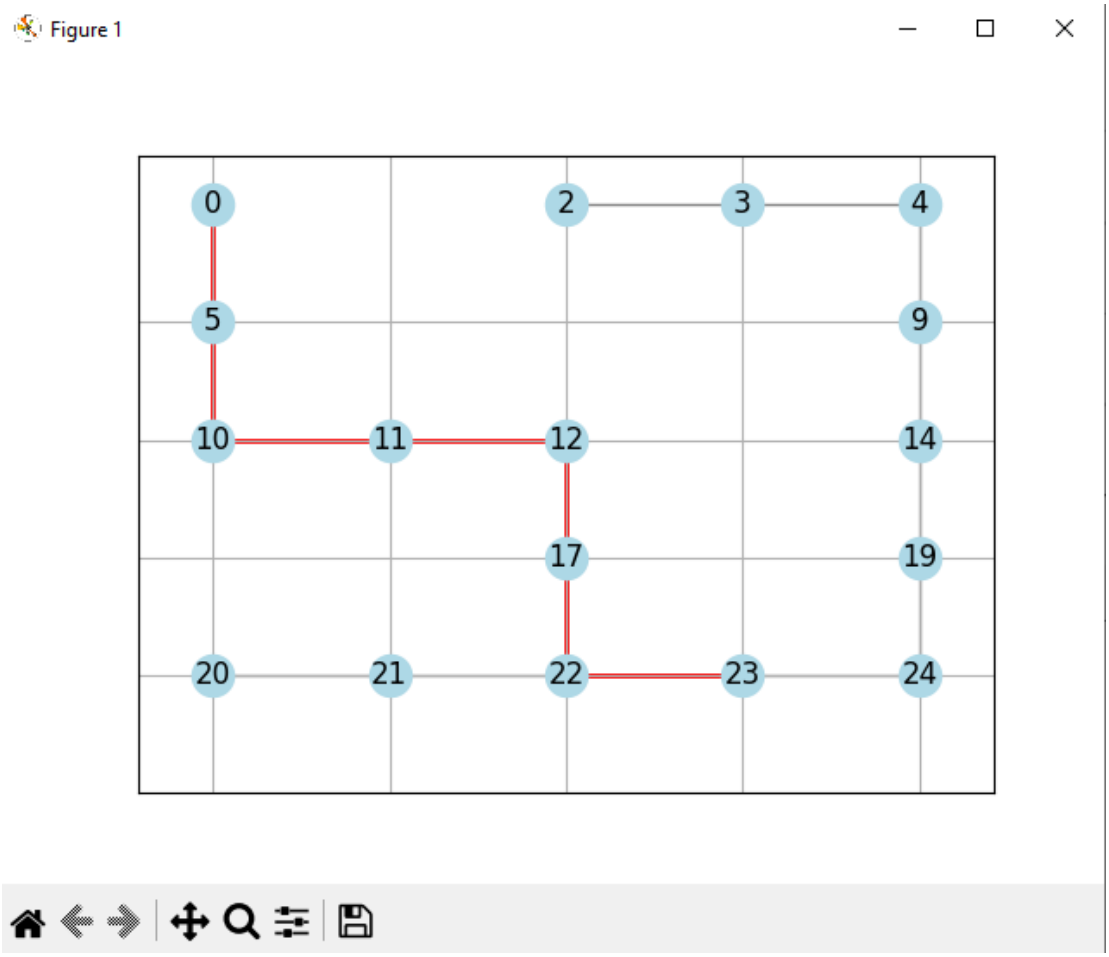
Код

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

def draw_graph(graph, maze_matrix, path=None):
    # Встановлення позицій вузлів графа відповідно до їхніх координат у матриці
    pos = dict()
    rows = len(maze_matrix)
    cols = len(maze_matrix[0])
    for i in range(rows):
        for j in range(cols):
            node = i * cols + j
            pos[node] = (j, -i)  # Використання від'ємних значень для правильного
відображення графа
    # Візуалізація вузлів та ребер графа
    nx.draw_networkx_nodes(graph, pos, node_color='lightblue')
    nx.draw_networkx_labels(graph, pos)
    nx.draw_networkx_edges(graph, pos, edge_color='gray')
    if path:
        edges = list(zip(path, path[1:]))
        nx.draw_networkx_edges(graph, pos, edgelist=edges, edge_color='red',
width=2.0)
    plt.xticks(range(cols))
    plt.yticks(range(-rows, 0))
    plt.grid(visible=True)
    plt.show()

def find_shortest_path(graph, start, end):
    # Алгоритм пошуку в ширину для знаходження найкоротшого шляху
    queue = [(start, [start])]
    while queue:
        (vertex, path) = queue.pop(0)
        for next_vertex in graph[vertex]:
            if next_vertex == end:
                return path + [next_vertex]
            else:
                queue.append((next_vertex, path + [next_vertex]))
    return None

def create_graph_from_maze(maze):
    rows = len(maze)
```

```
    cols = len(maze[0])
    graph = nx.Graph()
    for i in range(rows):
        for j in range(cols):
            if maze[i][j] != 1:
                node = i * cols + j
                graph.add_node(node)
                if maze[i][j] == 10:
                    end_node = node
                if i > 0 and maze[i - 1][j] != 1:
                    graph.add_edge(node, (i - 1) * cols + j)
                if i < rows - 1 and maze[i + 1][j] != 1:
                    graph.add_edge(node, (i + 1) * cols + j)
                if j > 0 and maze[i][j - 1] != 1:
                    graph.add_edge(node, i * cols + (j - 1))
                if j < cols - 1 and maze[i][j + 1] != 1:
                    graph.add_edge(node, i * cols + (j + 1))
    return graph, end_node

# Приклад лабіринту у вигляді матриці
maze_matrix = [
    [0, 1, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 0],
    [1, 1, 0, 1, 0],
    [0, 0, 0, 10, 0]
]

# Створення графа з матриці лабіринту
maze_graph, exit_node = create_graph_from_maze(maze_matrix)

# Візуалізація графа лабіринту
draw_graph(maze_graph, maze_matrix)

# Знаходження найкоротшого шляху
start_node = 0
shortest_path = find_shortest_path(maze_graph, start_node, exit_node)

if shortest_path:
    print(f"Найкоротший шлях: {shortest_path}")
    draw_graph(maze_graph, maze_matrix, shortest_path)
else:
    print("Шляху немає.")
```

Результат

```
maze_matrix = [
    [0, 1, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 0],
    [1, 1, 0, 1, 0],
    [0, 0, 0, 10, 0]
]
```



```
maze_matrix = [
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 1, 10, 0]
]
```