Дискретні структури ІПЗ-21-2

Губарєв Ростислав Вадимович

**Лабораторна робота №12.1**

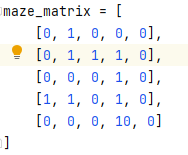
Вихід із лабіринту.

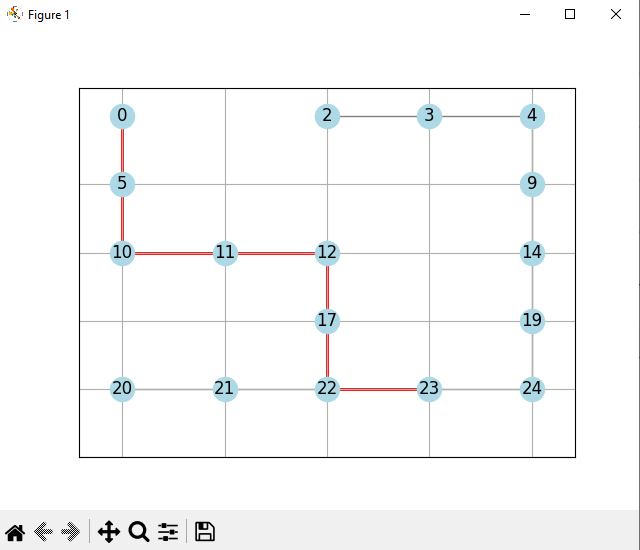
Поле лабіринту було розбито на клітини, після чого в прямокутну матрицю NxM було занесено інформацію про кожну клітину: 0, якщо переміщення через клітину можливе, і 1, якщо ні.  
Значенням "10" позначено вихід із лабіринту. На цій карті задано початкове положення гравця у вигляді координат клітини, де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

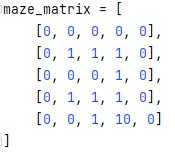
Уявіть лабіринт у вигляді графа і візуалізуйте його. Знайдіть найкоротший шлях гравця до виходу з лабіринту і намалюйте цей шлях на графі.  
Якщо таких шляхів кілька - виведіть будь-який із них. Якщо шляху немає - виведіть повідомлення про це.

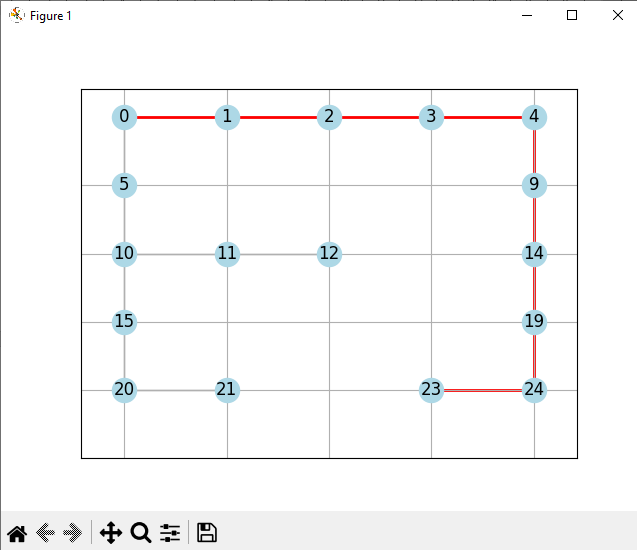
|  |
| --- |
| **Код**  import networkx as nx import matplotlib.pyplot as plt  def draw\_graph(graph, maze\_matrix, path=None):  *# Встановлення позицій вузлів графа відповідно до їхніх координат у матриці* pos = dict()  rows = len(maze\_matrix)  cols = len(maze\_matrix[0])  for i in range(rows):  for 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**

****

****

****

****