|  |
| --- |
| 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 chẳn 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.

Diagram

Description automatically generated

1. *(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() |

1. *(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: |

1. *(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.

1. (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 |

1. (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() |

1. (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) |