Họ và tên: Trần Đăng Khoa

MSV: 20212609

Câu 1:

1. Khái niệm về Trí tuệ Nhân tạo

Trí tuệ nhân tạo là khả năng của máy tính hoặc hệ thống máy tính có thể mô phỏng trí thông minh của con người thông qua việc học từ dữ liệu, nhận diện mẫu, giải quyết vấn đề và đưa ra quyết định. AI được chia thành ba loại chính:

- AI hẹp (Narrow AI): Chuyên xử lý các tác vụ cụ thể như nhận diện khuôn mặt, dịch ngôn ngữ, dự đoán thời tiết.
- AI tổng quát (General AI): Có khả năng thực hiện bất kỳ nhiệm vụ trí tuệ nào mà con người có thể làm. Hiện tại, General AI vẫn chưa được phát triển.
- AI tự nhận thức (Self-aware AI): Có thể hiểu được cảm xúc, ý thức và có khả năng tự nhận thức. Đây là một khái niệm lý thuyết và chưa tồn tại.

2. Lĩnh vực nghiên cứu của Trí tuệ Nhân tạo

Các lĩnh vực nghiên cứu chính trong AI bao gồm:

- Machine Learning (ML): Tập trung vào việc phát triển các thuật toán để máy tính có thể học từ dữ liệu mà không cần lập trình cụ thể. ML bao gồm các kỹ thuật như học có giám sát (supervised learning), học không giám sát (unsupervised learning), và học tăng cường (reinforcement learning).
- Deep Learning: Một nhánh của ML sử dụng các mạng nơ-ron nhân tạo với nhiều lớp (deep neural networks) để học từ dữ liệu phức tạp, thường được sử dụng trong nhận diện hình ảnh và xử lý ngôn ngữ tự nhiên.
- Xử lý Ngôn ngữ Tự nhiên (Natural Language Processing - NLP): Nghiên cứu cách để máy tính có thể hiểu, phân tích và tạo ra ngôn ngữ con người.
- Computer Vision: Cho phép máy tính nhận diện và phân tích thông tin từ hình ảnh hoặc video.

- Robotics**:** Liên quan đến việc thiết kế và lập trình các robot thông minh có khả năng tương tác với môi trường xung quanh.
- Expert Systems**:** Các hệ thống có thể mô phỏng khả năng ra quyết định của con người dựa trên kiến thức chuyên môn.

3. Ứng dụng của Trí tuệ Nhân tạo

AI đã và đang được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau, bao gồm:

- Y tế**:** Chẩn đoán bệnh, phân tích hình ảnh y khoa, dự đoán tiến trình bệnh, và hỗ trợ quản lý dược phẩm.
- Tài chính**:** Phân tích rủi ro, phát hiện gian lận, dự báo thị trường, và tối ưu hóa danh mục đầu tư.
- Giáo dục**:** Cá nhân hóa chương trình học, tự động chấm điểm, và hỗ trợ học tập thông qua các trợ lý ảo.
- Giao thông vận tải**:** Xe tự lái, tối ưu hóa tuyến đường, và hệ thống quản lý giao thông thông minh.
- Thương mại điện tử: Đề xuất sản phẩm, phân tích hành vi người dùng, và hỗ trợ khách hàng tự động.
- Sản xuất: Tự động hóa quy trình sản xuất, bảo trì dự đoán, và quản lý chuỗi cung ứng.
- Giải trí**:** Trò chơi video, tạo nội dung tự động, và phân tích cảm xúc khán giả.

Câu 2:

1. Khái niệm
   - Thuật toán A* là một thuật toán tìm kiếm heuristic (tìm kiếm theo hướng dẫn) giúp tìm đường đi ngắn nhất từ điểm xuất phát đến điểm đích trên một đồ thị. Thuật toán này sử dụng cả chi phí đã đi từ điểm xuất phát đến một điểm hiện tại, và chi phí ước lượng từ điểm hiện tại đến đích, để xác định đường đi tối ưu
   - Greedy Best-First Search sử dụng một hàm heuristic h(n) để đánh giá chi phí ước lượng từ điểm hiện tại đến đích. GBFS chỉ quan tâm đến hàm heuristic h(n) mà không tính đến chi phí thực tế g(n) từ điểm bắt đầu đến điểm hiện tại. Điều này làm cho GBFS hoạt động nhanh chóng, nhưng không đảm bảo sẽ tìm ra đường đi ngắn nhất.

2. Cài đặt thuật toán

```python
# Thuật toán A*


import time
import networkx as nx
import heapq
import random

def a_star_search(G, start, goal, heuristic):
    open_set = []
    heapq.heappush(open_set, (0 + heuristic[start], start))

    g_costs = {node: float('inf') for node in G.nodes()}
    g_costs[start] = 0
    came_from = {start: None}

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

        if current == goal:
            path = []
            while current is not None:
                path.append(current)
                current = came_from[current]
            path.reverse()
            return path

        for neighbor in G.neighbors(current):
            weight = G[current][neighbor].get('weight', 1)
            tentative_g_cost = g_costs[current] + weight

            if tentative_g_cost < g_costs[neighbor]:
                came_from[neighbor] = current
                g_costs[neighbor] = tentative_g_cost
                f_cost = tentative_g_cost + heuristic[neighbor]
                heapq.heappush(open_set, (f_cost, neighbor))

    return None

def example_heuristic(node):
    return random.randint(1, 10)
```

```python
def read_graph_with_default_weights(file_path):
    G = nx.Graph()
    with open(file_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            if len(parts) < 2:
                continue

            node1, node2 = parts[:2]

            try:
                weight = float(parts[2]) if len(parts) > 2 else 1.0
            except ValueError:
                weight = 1.0

            G.add_edge(node1, node2, weight=weight)

    return G
start_time = time.time()

G = read_graph_with_default_weights("random_graph.edgelist")

start_node = list(G.nodes())[0]
goal_node = list(G.nodes())[-1]

heuristic = {node: example_heuristic(node) for node in G.nodes()}


path = a_star_search(G, start_node, goal_node, heuristic)
end_time = time.time()
elapsed_time = end_time - start_time
print("Path from start to goal:")
print(path)

print(f"Elapsed time: {elapsed_time:.2f} seconds")
```

```python
# Thuật toán GBFS

import time
import networkx as nx
from queue import PriorityQueue

def read_graph_from_file(file_path):
    G = nx.read_edgelist(file_path, nodetype=int)
    return G

def greedy_bfs(graph, start, goal, heuristic):
    pq = PriorityQueue()
    pq.put((0, start))
    visited = set()
    path = []

    while not pq.empty():
        _, current_node = pq.get()

        path.append(current_node)

        if current_node == goal:
            return path

        visited.add(current_node)

        for neighbor in graph.neighbors(current_node):
            if neighbor not in visited:
                priority = heuristic(neighbor, goal)
                pq.put((priority, neighbor))

    return None

def heuristic(node, goal):
    return abs(node - goal)

if __name__ == "__main__":
    start_time = time.time()
    file_path = "random_graph.edgelist"
    G = read_graph_from_file(file_path)

    start_node = 0
    goal_node = 10

    result_path = greedy_bfs(G, start_node, goal_node, heuristic)
```

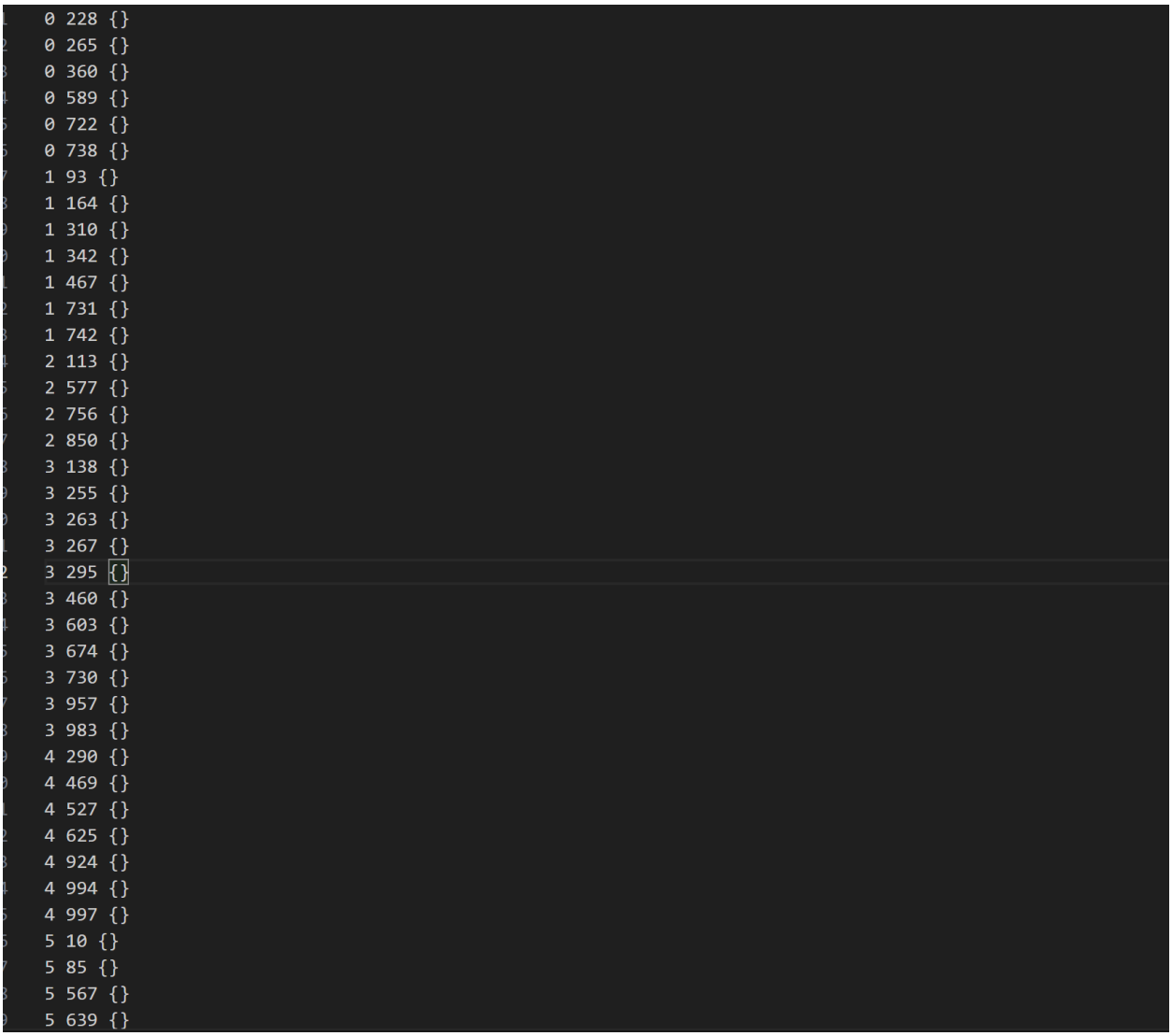
```
    end_time = time.time()
    elapsed_time = end_time - start_time
    if result_path:
        print("Path found:", result_path)
    else:
        print("No path found between {} and {}".format(start_node, goal_node))

    print(f"Elapsed time: {elapsed_time:.2f} seconds")
```

3. Kết quả và đánh giá

Lần 1:

Data:



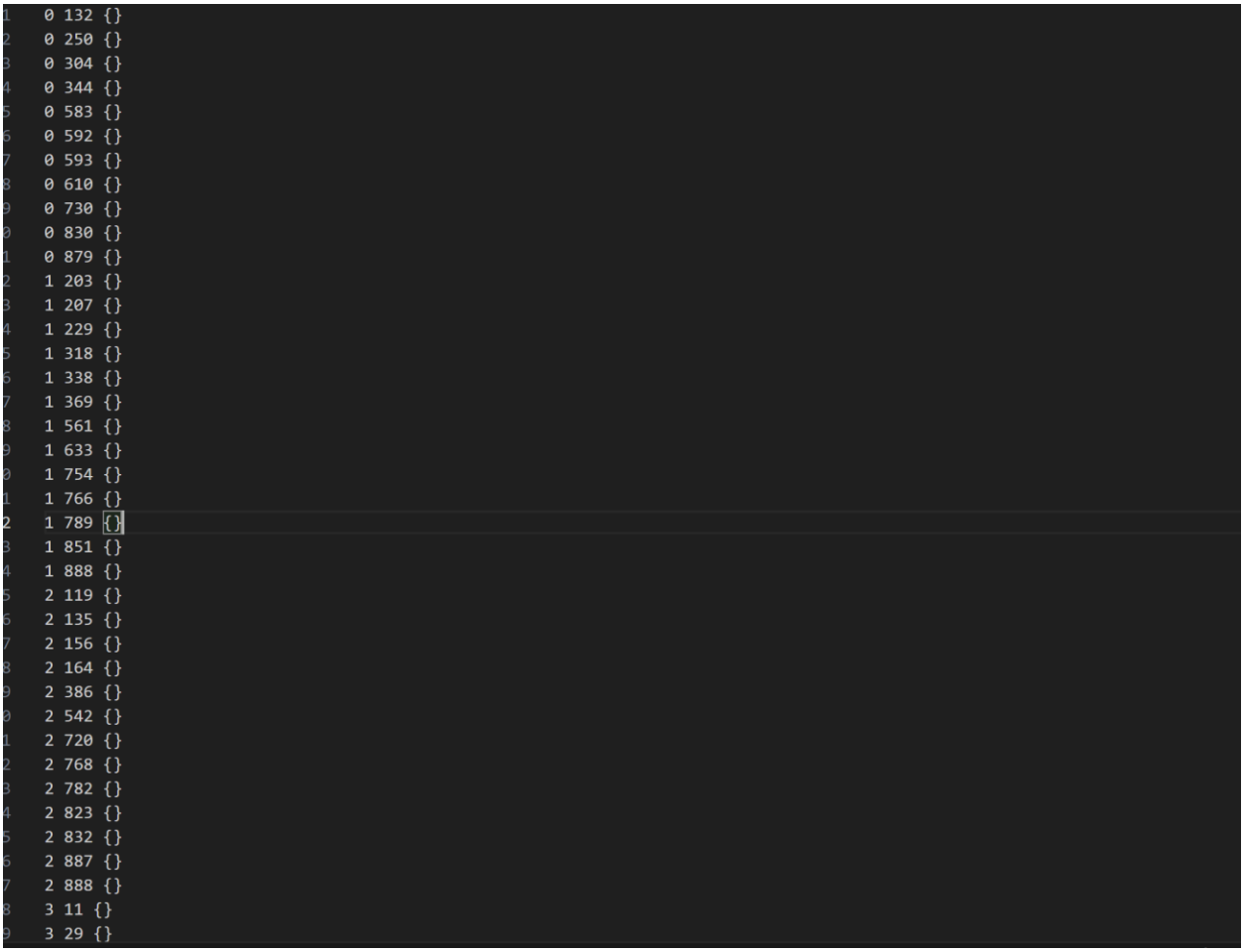
```
0 228 {}
0 265 {}
0 360 {}
0 589 {}
0 722 {}
0 738 {}
1 93 {}
1 164 {}
1 310 {}
1 342 {}
1 467 {}
1 731 {}
1 742 {}
2 113 {}
2 577 {}
2 756 {}
2 850 {}
3 138 {}
3 255 {}
3 263 {}
3 267 {}
3 295 {}
3 460 {}
3 603 {}
3 674 {}
3 730 {}
3 957 {}
3 983 {}
4 290 {}
4 469 {}
4 527 {}
4 625 {}
4 924 {}
4 994 {}
4 997 {}
5 10 {}
5 85 {}
5 567 {}
5 639 {}
```

Thời gian chạy:

- A*: 0.05s
- GBFS: 0.08s

Lần 2:

Data:

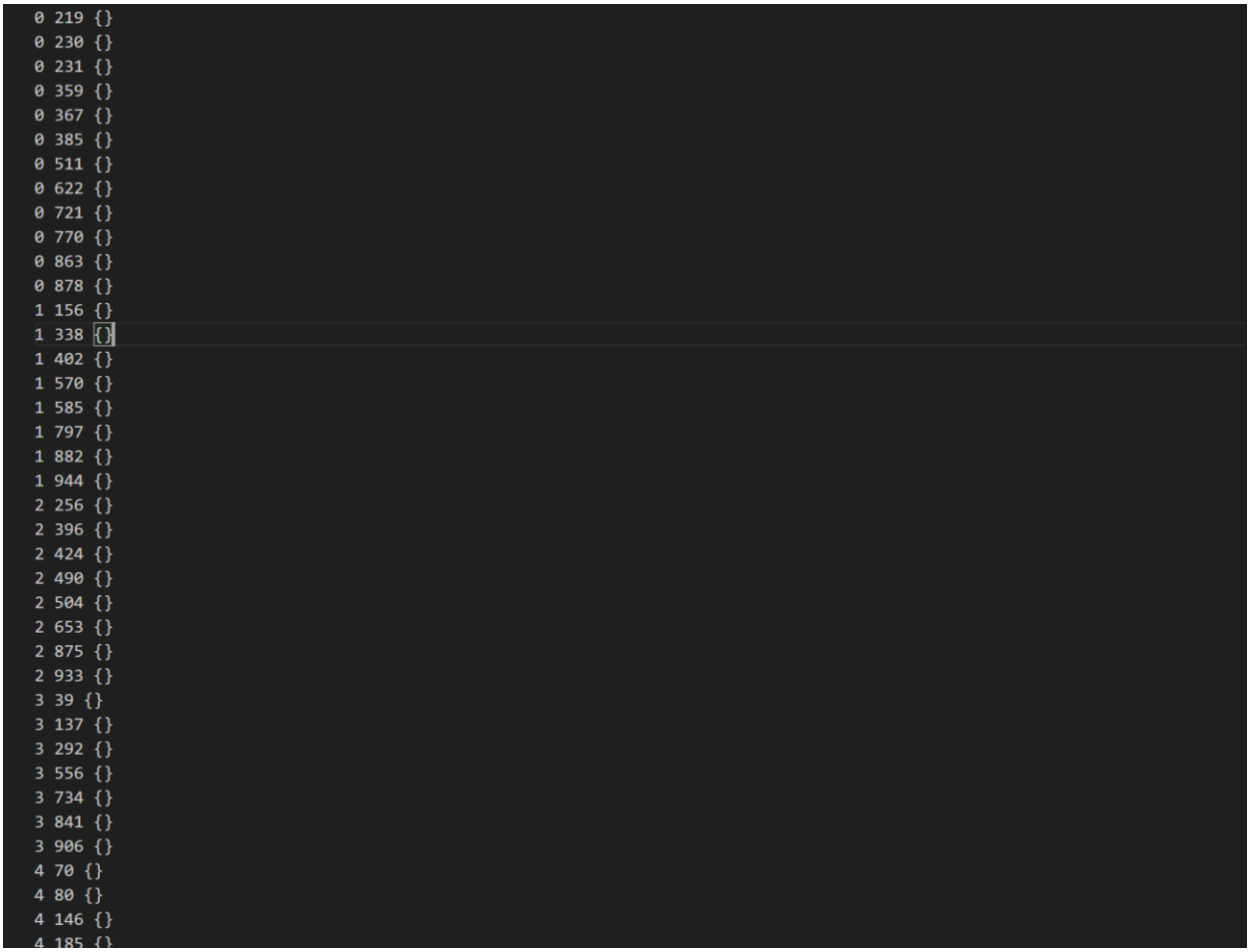


Thời gian chạy:

- A*: 0.08
- GBFS: 0.14s

Lần 3:

Data:



Thời gian chạy:

- A*: 0.02
- GBFS: 0.04s

Kết luận: Sau 3 lần chạy với 3 bộ dữ liệu khác nhau ta có thể thấy A* là 1 thuật toán tối ưu về thời gian, thời gian chạy của nó nhanh và trả ra kết quả chính xác