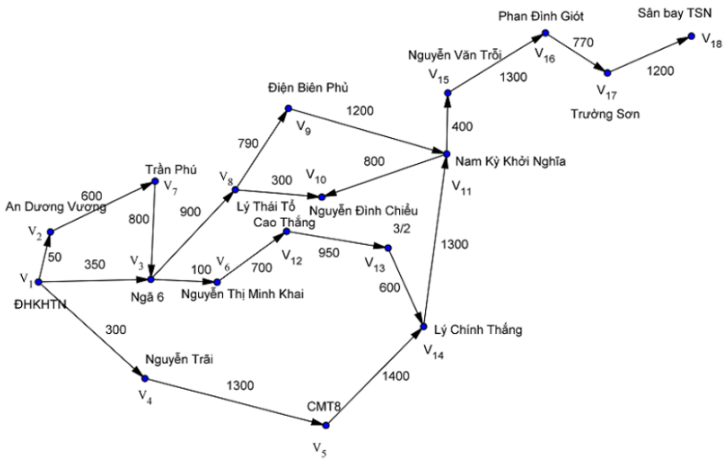
Họ và tên: Phan Hồng Trâm

MSSV: 211110414

# Báo cáo trình bày

# Thực hành Nhập môn trí tuệ nhân tạo tuần 1

## ❖ Chạy tay thuật toán BFS, DFS, UCS




- **Thuật toán BFS:**

1. L = [$V_1$]
2. Node = $V_1$, L = [$V_2$, $V_3$, $V_4$], father [$V_2$, $V_3$, $V_4$] = $V_1$
3. Node = $V_2$, L = [$V_3$, $V_4$, $V_7$], father [$V_7$] = $V_2$
4. Node = $V_3$, L = [$V_4$, $V_7$, $V_6$, $V_8$], father [$V_6$, $V_8$] = $V_3$
5. Node = $V_4$, L = [$V_7$, $V_6$, $V_8$, $V_5$], father [$V_5$] = $V_4$
6. Node = $V_7$, L = [$V_6$, $V_8$, $V_5$]
7. Node = $V_6$, L = [$V_8$, $V_5$, $V_{12}$], father [$V_{12}$] = $V_6$
8. Node = $V_8$, L = [$V_5$, $V_{12}$, $V_9$, $V_{10}$], father [$V_9$, $V_{10}$] = $V_8$
9. Node = $V_5$, L = [$V_{12}$, $V_9$, $V_{10}$, $V_{14}$], father [$V_{14}$] = $V_5$
10. Node = $V_{12}$, L = [$V_9$, $V_{10}$, $V_{14}$, $V_{13}$], father [$V_{13}$] = $V_{12}$
11. Node = $V_9$, L = [$V_{10}$, $V_{14}$, $V_{13}$, $V_{11}$], father [$V_{11}$] = $V_9$
12. Node = $V_{10}$, L = [$V_{14}$, $V_{13}$, $V_{11}$]
13. Node = $V_{14}$, L = [$V_{13}$, $V_{11}$]
14. Node = $V_{13}$, L = [$V_{11}$]
15. Node = $V_{11}$, L = [$V_{15}$], father [$V_{15}$] = $V_{11}$
16. Node = $V_{15}$, L = [$V_{16}$], father [$V_{16}$] = $V_{15}$
17. Node = $V_{16}$, L = [$V_{17}$], father [$V_{17}$] = $V_{16}$
18. Node = $V_{17}$, L = [$V_{18}$], father [$V_{18}$] = $V_{17}$

Vậy đường đi ngắn nhất từ $V_1$ đến $V_{18}$ là: $V_1 \rightarrow V_3 \rightarrow V_8 \rightarrow V_9 \rightarrow V_{11} \rightarrow V_{15} \rightarrow V_{16} \rightarrow V_{17} \rightarrow V_{18}$

- **Thuật toán DFS:**

1. L = [$V_1$]
2. Node = $V_1$, L = [$V_4$, $V_3$, $V_2$], father [$V_4$, $V_3$, $V_2$] = $V_1$
3. Node = $V_4$, L = [$V_5$, $V_3$, $V_2$], father [$V_5$] = $V_4$
4. Node = $V_5$, L = [$V_{14}$, $V_3$, $V_2$], father [$V_{14}$] = $V_5$
5. Node = $V_{14}$, L = [$V_{11}$, $V_3$, $V_2$], father [$V_{11}$] = $V_{14}$
6. Node = $V_{11}$, L = [$V_{15}$, $V_3$, $V_2$], father [$V_{15}$] = $V_{11}$
7. Node = $V_{15}$, L = [$V_{16}$, $V_3$, $V_2$], father [$V_{16}$] = $V_{15}$
8. Node = $V_{16}$, L = [$V_{17}$, $V_3$, $V_2$], father [$V_{17}$] = $V_{16}$
9. Node = $V_{17}$, L = [$V_{18}$, $V_3$, $V_2$], father [$V_{18}$] = $V_{17}$

Vậy đường đi ngắn nhất từ $V_1$ đến $V_{18}$ là: $V_1 \rightarrow V_4 \rightarrow V_5 \rightarrow V_{14} \rightarrow V_{11} \rightarrow V_{15} \rightarrow V_{16} \rightarrow V_{17} \rightarrow V_{18}$

- **Thuật toán UCS:**

1. PQ = ($V_1$, 0)
2. PQ = ($V_2$, 50), ($V_4$, 300), ($V_3$, 350)
3. PQ = ($V_4$, 300), ($V_3$, 350), ($V_7$, 650)
4. PQ = ($V_3$, 350), ($V_7$, 650), ($V_5$, 1600)
5. PQ = ($V_6$, 450), ($V_7$, 650), ($V_8$, 1250), ($V_5$, 1600)
6. PQ = ($V_7$, 650), ($V_{12}$, 1150), ($V_8$, 1250), ($V_5$, 1600)
7. PQ = ($V_{12}$, 1150), ($V_8$, 1250), ($V_5$, 1600)
8. PQ = ($V_8$, 1250), ($V_5$, 1600), ($V_{13}$, 2100)
9. PQ = ($V_{10}$, 1550), ($V_5$, 1600), ($V_9$, 2040), ($V_{13}$, 2100)
10. PQ = ($V_5$, 1600), ($V_9$, 2040), ($V_{13}$, 2100)
11. PQ = ($V_9$, 2040), ($V_{13}$, 2100), ($V_{14}$, 3000)
12. PQ = ($V_{13}$, 2100), ($V_{14}$, 3000), ($V_{11}$, 3240)
13. PQ = ($V_{14}$, 2700), ($V_{11}$, 3240)
14. PQ = ($V_{11}$, 3240)
15. PQ = ($V_{15}$, 3640)
16. PQ = ($V_{16}$, 4940)
17. PQ = ($V_{17}$, 5710)
18. PQ = ($V_{18}$, 6910)

Vậy đường đi ngắn nhất từ $V_1$ đến $V_{18}$ là: $V_1 \rightarrow V_3 \rightarrow V_8 \rightarrow V_9 \rightarrow V_{11} \rightarrow V_{15} \rightarrow V_{16} \rightarrow V_{17} \rightarrow V_{18}$

Với chi phí đường đi là 6910.

❖ **Cài đặt và thực thi chương trình. Nếu chương trình bị báo lỗi thì lỗi ở dòng nào và sửa lại như thế nào?**

- Chương trình không bị báo lỗi và hoạt động bình thường.

❖ **Kiểm tra tính đúng đắn của các thuật toán đã cho sẵn code như trên. Nếu chưa đúng thì em sửa lại như thế nào cho phù hợp?**

- Về mặt cú pháp và logic, code không bị lỗi và hoạt động bình thường.
- Tuy nhiên, đoạn code dùng cấu trúc dữ liệu `list()` cho biến **`visited`** để lưu trữ các node đã khám phá (explored set). Theo em, chúng ta không nên dùng `list()` cho biến **`visited`**, điều này dẫn đến lãng phí bộ nhớ và gây lặp lại mã. Dưới đây là những dòng code lặp lại trong mảng **`visited`**:

```
while True:
    if frontier.empty():
        raise Exception("No way Exception")
    current_node = frontier.get()
    visited.append(current_node)

    # Kiểm tra current_node có là end hay không
    if current_node == end:
        path_found = True
        break

    for node in graph[current_node]:
        if node not in visited:
            frontier.put(node)
            parent[node] = current_node
            visited.append(node)
```

BFS

```
while True:
    if frontier == []:
        raise Exception("No way Exception")
    current_node = frontier.pop()
    visited.append(current_node)

    # Kiểm tra current_node có là end hay không
    if current_node == end:
        path_found = True
        break

    for node in graph[current_node]:
        if node not in visited:
            frontier.append(node)
            parent[node] = current_node
            visited.append(node)
```

DFS

```
while True:
    if frontier.empty():
        raise Exception("No way Exception")
    current_w,current_node = frontier.get()
    visited.append(current_node)

    # Kiểm tra current_node có là end hay không
    if current_node == end:
        path_found = True
        break

    for nodei in graph[current_node]:
        node, weight = nodei
        if node not in visited:
            frontier.put((current_w + weight, node))
            parent[node] = current_node
            visited.append(node)
```

UCS

➔ **Giải pháp**: Ta nên dùng cấu trúc dữ liệu **`dict()`** (giống với biến **`parent`**).

- Khởi tạo biến **`visited`** ta dùng `dict()`và tạo value là `False`, khi node đó đã được explored thì thiết lập lại là `True` (Truyền tham số **`size`** vào mỗi hàm tìm kiếm để thuận lợi cho việc thiết lập biến **`visited`**, cụ thể BFS và DFS ta truyền **`size_1`** còn UCS ta truyền **`size_2`**):

```
visited = {i: False for i in range(size)}
```

- Bỏ đi những dòng code không cần thiết:

```
visited.append(node)
```

- Thay đổi cách mở rộng node thông qua **`dict()`**:

```
for node in graph[current_node]:
  if visited[node] == False:
    …
    visited[node] = True
```

✓ **Lợi ích:** Làm giảm không gian lưu trữ của **`visited`**, giúp code ngắn gọn và tối ưu bộ nhớ hơn.

❖ **Nhận xét kết quả chạy tay với kết quả chạy máy?**

- Với **BFS**: Kết quả chạy tay giống với kết quả chạy máy.
- Với **DFS**: Có 3 trường hợp khi chạy tay (Ở bước 2, có 3 node [V4, V3, V2], tùy thuộc vào ta chọn node nào vào trước). Trường hợp ngắn nhất là ở bước 2 ta chọn node V4 trước, và kết quả chạy tay giống với kết quả chạy máy.
- Với **UCS**: Kết quả chạy tay ra giống với kết quả chạy máy.