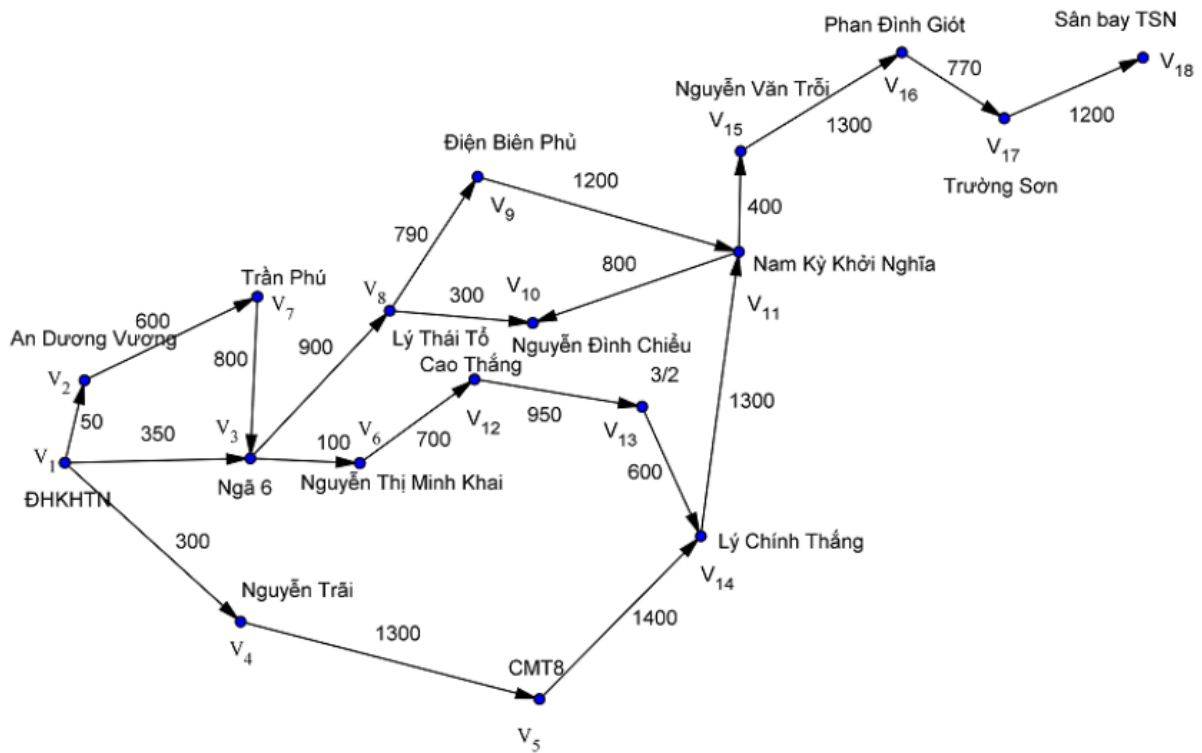


THỰC HÀNH MÔN TRÍ TUỆ NHÂN TẠO_TUẦN 1

Bài toán: Cho đồ thị như hình vẽ bên dưới



Tìm đường đi ngắn nhất từ trường Đại học Khoa học Tự nhiên (V_1) tới sân bay Tân Sơn Nhất (V_{18}) dùng các thuật toán sau:

1. BFS
2. DFS
3. UCS

HƯỚNG DẪN

1. Dữ liệu đầu vào và đầu ra:

- Dữ liệu đầu vào:

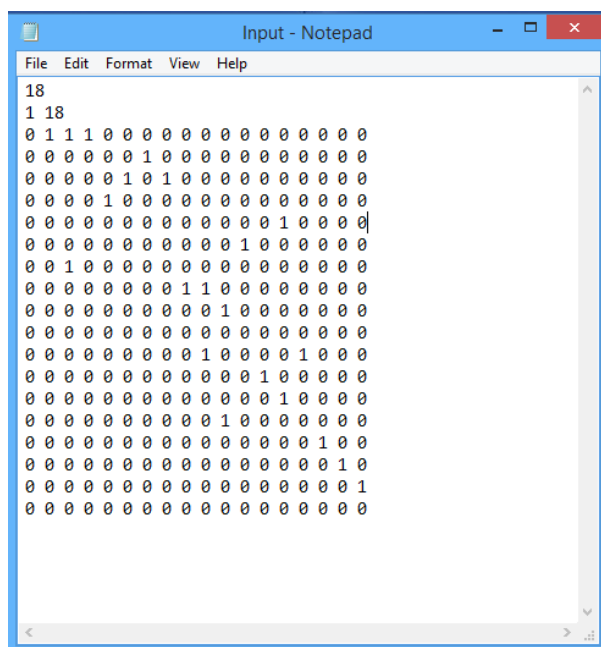
❖ Dòng 1: Số node trên đồ thị

❖ Dòng 2: node xuất phát và node đích

❖ Những dòng tiếp theo: ma trận kề M của đồ thị với quy ước:

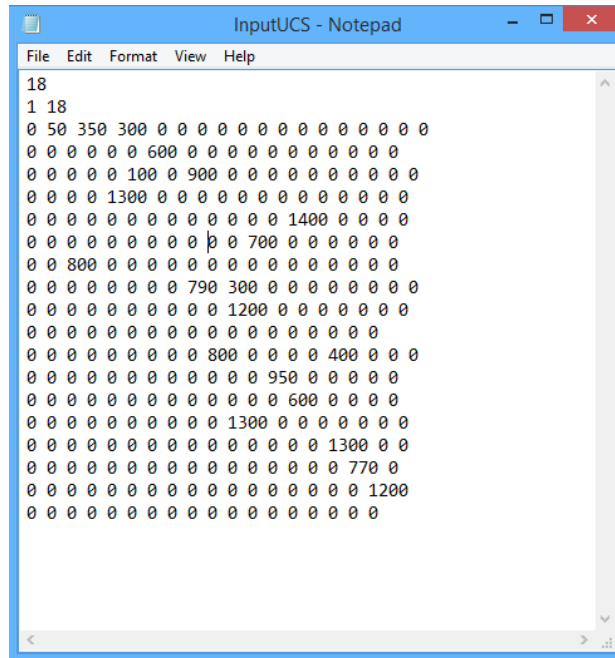
- $M[i][j] = 1$: có đường nối trực tiếp từ i tới j ($M[i][j] = w$: có đường nối trực tiếp từ i đến j với chi phí là w ($w > 0$) cho thuật toán UCS).
- $M[i][j] = 0$: không có đường nối trực tiếp từ i tới j

🗺 Dữ liệu cho BFS và DFS



```
Input - Notepad
File Edit Format View Help
18
1 18
0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

🗺 Dữ liệu cho UCS



⇒ lưu 2 file như hình với định dạng .txt ⇒ đọc dữ liệu từ file .txt

🚦 Đọc dữ liệu từ file trong Python:

open(file, mode)

trong đó:

- **file**: đường dẫn và tên của file
- **mode**:
 - “r” – mở file để đọc (read)
 - “w” – mở file để viết (write)

read(): trả về 1 chuỗi

readline(): trả về 1 dòng

readlines(): trả về danh sách các dòng

write(): viết 1 chuỗi vào file

writelines(): viết danh sách các chuỗi vào file

close(): đóng file

string.split(separator, maxsplit): chuyển chuỗi thành list (separator: dấu ngăn cách để tách chuỗi (mặc định là khoảng trắng), maxsplit: mặc định là -1 (tất cả các lần xuất hiện))

⇒ **xây dựng graph ???**

- Đầu ra:
 - Nếu tồn tại đường đi: xuất ra màn hình thứ tự đường đi từ V_1 tới V_{18} .
 - Nếu không tồn tại đường đi: thông báo không có đường đi.

2. Cài đặt BFS

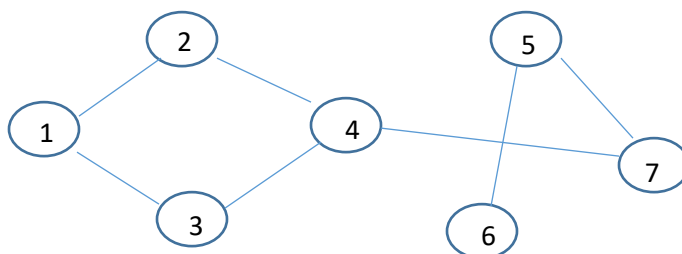
- ❖ **Ý tưởng:** tại mỗi bước chọn trạng thái để phát triển là trạng thái được sinh ra trước các trạng thái chờ phát triển khác. Danh sách L được sử lý như hàng đợi (queue).

Procedure *Breadth_Search*

Begin

1. Khởi tạo danh sách L chứa trạng thái ban đầu;
 2. **While** (1)
 - 2.1 **if** L rỗng **then**
 - {
 - Thông báo tìm kiếm thất bại;
 - stop**;
 - }
 - 2.2 Loại trạng thái u ở đầu danh sách L;
 - 2.3 **if** u là trạng thái kết thúc **then**
 - {
 - Thông báo tìm kiếm thành công;
 - stop**;
 - }
 - 2.4 Lấy các trạng thái v kề với u và thêm vào cuối danh sách L;
 - for** mỗi trạng thái v kề u **do**
 - $\text{father}(v) = u$;
- end**

- ❖ Ví dụ: Tìm đường đi từ đỉnh 1 tới đỉnh 7



- Các bước thực hiện của thuật toán BFS:

1. $L = [1]$
2. Node = 1, $L = [2, 3]$, $\text{father}[2, 3] = 1$
3. Node = 2, $L = [3, 4]$, $\text{father}[4] = 2$
4. Node = 3, $L = [4]$ (đỉnh 4 kề với đỉnh 3 nhưng đã tồn tại trong L nên không thêm vào), $\text{father}[7] = 4$.

⇒ Đường đi từ đỉnh 1 tới đỉnh 7 là: $1 \rightarrow 2 \rightarrow 4 \rightarrow 7$ hoặc $1 \rightarrow 3 \rightarrow 4 \rightarrow 7$.

❖ Cài đặt:

```
from queue import Queue, PriorityQueue

def bfs(graph, start, end):
    frontier = Queue()
    frontier.put(start)
    explored = []

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

        # Check if node is goal-node
        if current_node == end:
            return

        for node in graph[current_node]:
            if node not in explored:
                frontier.put(node)
```

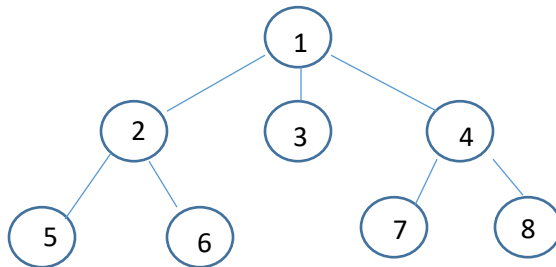
- **Thuật toán DFS:** Ý tưởng: tại mỗi bước trạng thái được chọn để phát triển là trạng thái được sinh ra sau cùng trong số các trạng thái chờ phát triển. Danh sách L được xử lý như ngăn xếp (stack).

Procedure *Depth_Search*

Begin

```
3. Khởi tạo danh sách L chứa trạng thái ban đầu;
4. While (1)
    2.5 if L rỗng then
        {
            Thông báo tìm kiếm thất bại;
            stop;
        }
    2.6 Loại trạng thái u ở đầu danh sách L;
    2.7 if u là trạng thái kết thúc then
        {
            Thông báo tìm kiếm thành công;
            stop;
        }
    2.8 Lấy các trạng thái v kề với u và thêm vào đầu danh sách L;
        for mỗi trạng thái v kề u do
            father(v) = u;
end
```

❖ Ví dụ:



Tìm đường đi từ đỉnh 1 tới đỉnh 6.

- Các bước thực hiện của thuật toán DFS:
 1. $L = [1]$
 2. Node = 1, $L = [4, 3, 2]$, $\text{father}[4, 3, 2] = 1$
 3. Node = 4, $L = [8, 7, 3, 2]$, $\text{father}[8, 7] = 4$

4. Node = 8, L = [7, 3, 2]
5. Node = 7, L = [3, 2]
6. Node = 3, L = [2]
7. Node = 2, L = [6, 5], father[6, 5] = 2
8. Node = 6

⇒ Đường đi từ đỉnh 1 tới đỉnh 6 là: $1 \rightarrow 2 \rightarrow 6$.

❖ Cài đặt:

```
def dfs(graph, start, end):  
    frontier = [start, ]  
    explored = []  
  
    while True:  
        if len(frontier) == 0:  
            raise Exception("No way Exception")  
        current_node = frontier.pop()  
        explored.append(current_node)  
  
        if current_node == end:  
            return  
  
        for node in reversed(graph[current_node]):  
            if node not in explored:  
                frontier.append(node)
```

➤ Thuật toán UCS:

function Tìm_kiểm_UCS(bài_toán, ngăn_chứa) **return** lời_giải hoặc thất_bại.

ngăn_chứa \leftarrow Tạo_Hàng_Đội_Rỗng()

ngăn_chứa \leftarrow Thêm(TẠO_NÚT(Trạng_Thái_Đầu[bài_toán]), ngăn_chứa)

loop do

if Là_Rỗng(ngăn_chứa) **then return** thất_bại.

 nút \leftarrow Lấy_Chi_phi_Nhỏ_nhất(ngăn_chứa)

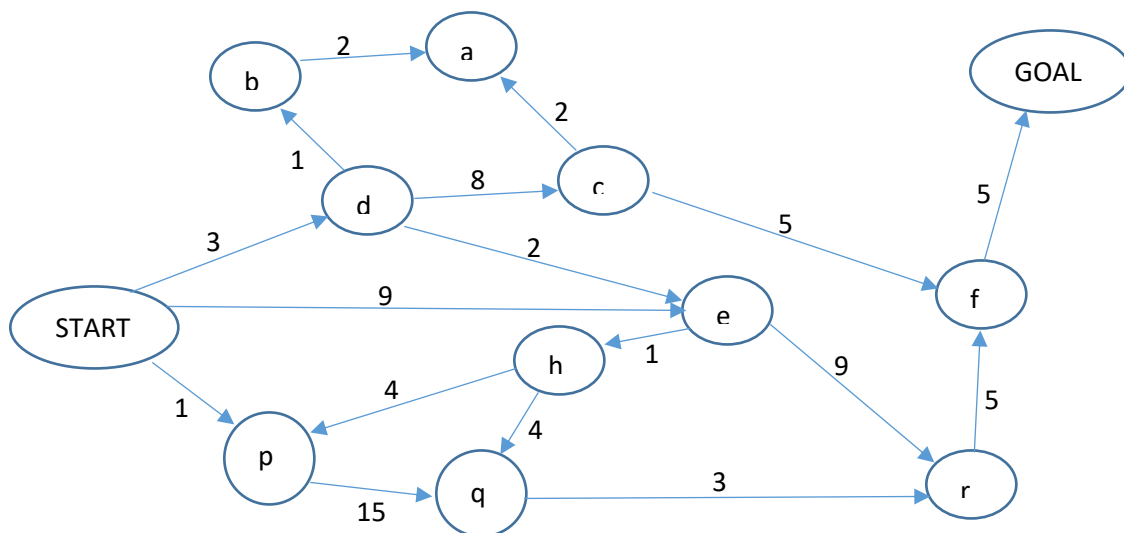
if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng.

then return Lời_giải(nút).

 lg \leftarrow Mở(nút, bài_toán) //lg tập các nút con mới

 ngăn_chứa \leftarrow Thêm_Tất_cả(lg, ngăn_chứa)

❖ Ví dụ: Cho đồ thị như hình



Các bước thực hiện của thuật toán Tìm_kiểm_UCS:

1. $PQ = \{(START, 0)\}$. (PQ là Priority Queue)
2. $PQ = \{(p, 1), (d, 3), (e, 9)\}$
3. $PQ = \{(d, 3), (e, 9), (q, 16)\}$
4. $PQ = \{(b, 4), (e, 5), (c, 11), (q, 16)\}$

5. PQ = {(e,5), (a,6), (c, 11), (q, 16)}
6. PQ = {(a,6), (h,6), (c,11), (r, 14), (q,16)}
7. PQ = {(h,6), (c,11), (r,14), (q,16)}
8. PQ = {(q,10), (c,11), (r,14)}
9. PQ = {(c,11), (r,13)}
10. PQ = {(r,13), (f, 16)}
11. PQ = {(f,16)}
12. PQ = {(GOAL, 21)}

⇒ Đường đi ngắn nhất từ START tới GOAL là: START → d → e
→ h → q → r → f → GOAL với chi phí là 21.

❖ Cài đặt:

```
def ucs(graph, start, end, weights=None):

    frontier = PriorityQueue()
    frontier.put((0, start))
    explored = []

    while True:
        if frontier.empty():
            raise Exception("No way Exception")

        ucs_w, current_node = frontier.get()
        explored.append(current_node)

        if current_node == end:
            return

        for node in graph[current_node]:
            if node not in explored:
                frontier.put((
                    ucs_w + ucs_weight(current_node, node, weights),
                    node
                ))
```