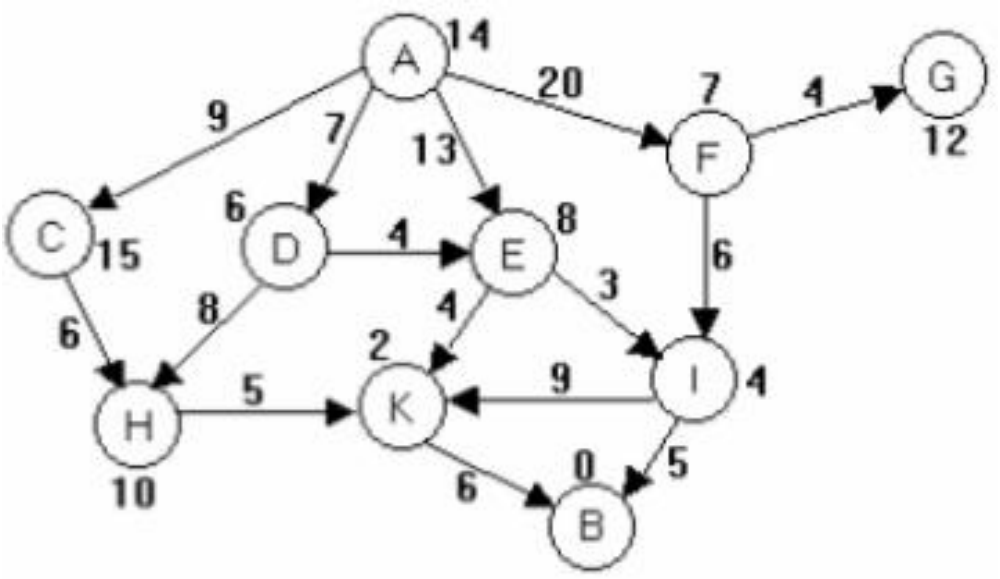
BÀI THỰC HÀNH – CSTTNT
BUỔI 4 - GIẢI THUẬT A*

Bài 1. Cho đồ thị sau:



và mã giả giải thuật A*

1. Open:={s}; Close:={};
2. Trong khi (Open khác rỗng)
   - 2.1 Chọn đỉnh p tốt nhất trong Open
   - 2.2 Nếu p là đỉnh kết thúc thì thoát.
   - 2.3 Di chuyển đỉnh p qua Close và tạo danh sách các đỉnh q có nối với p
     - 2.3.1 Nếu q có trong Open:
       - Nếu (**g(q) > g(p) +cost(p,q)**) thì — Phát hiện đường đi mới ngắn hơn
         - *g(q) = g(p) +cost(p,q)*
         - *f(q) = g(q) + h(q);*
         - *Nút trước của q là p;*
     - 2.3.2 Nếu q chưa có trong Open thì
       - *g(q) = g(p) + cost(p,q);*
       - *f(q) = g(q) + h(q);*
       - *Thêm q vào Open;*
       - *Nút trước q là p;*
     - 2.3.3 Nếu q có trong Close thì
       - Nếu (**g(q)>g(p)+cost(p,q)**) thì — Phát hiện đường đi mới ngắn hơn
         - *Di chuyển q vào Open;*
         - *Nút trước của q là p;*
3. Không tìm được.

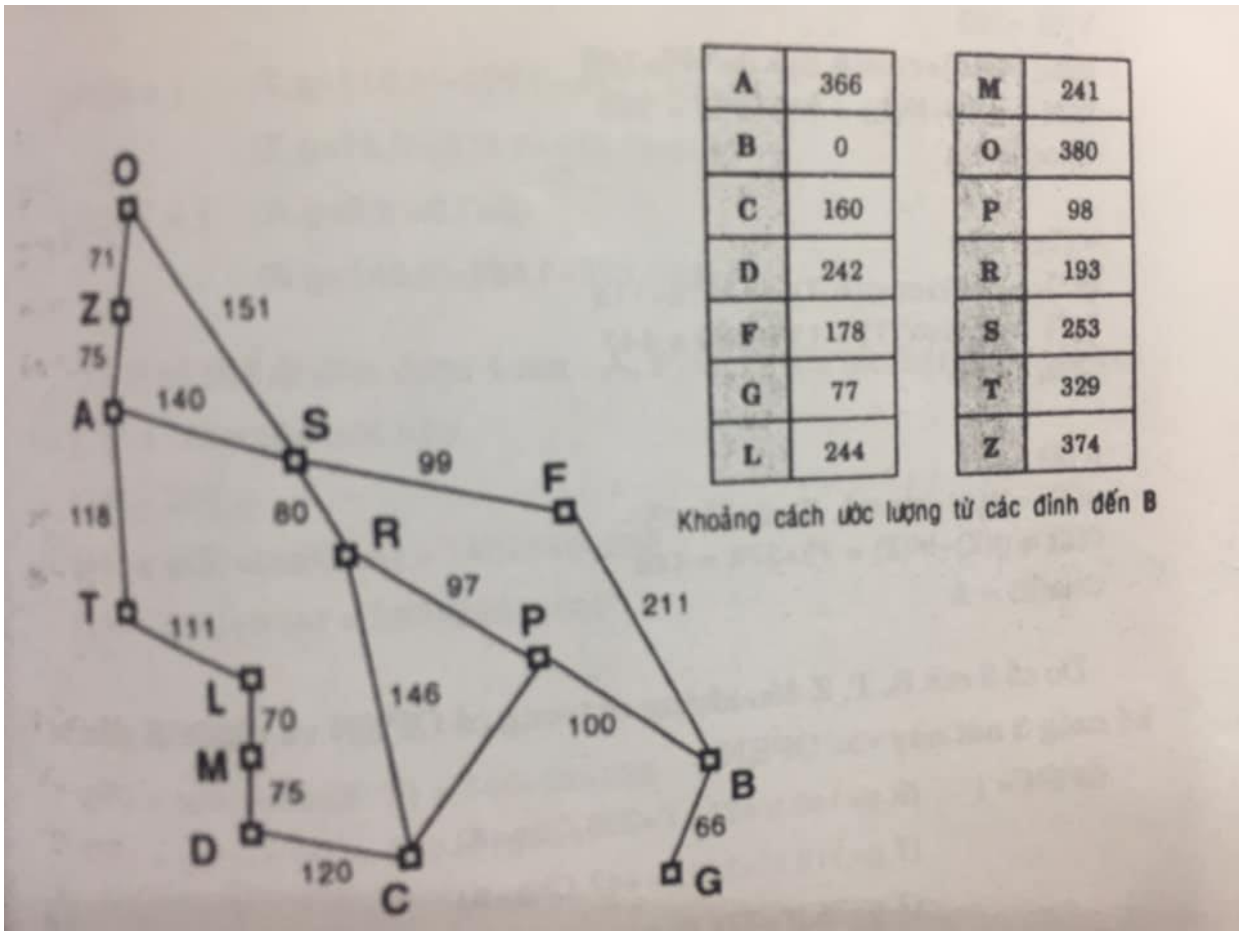
Yêu cầu: Hiện thực giải thuật A* để tìm đường đi từ **start_node** đến **end_node**

```
#Khai báo class Graph
class Graph:
    #constructor
    def __init__(self, adjacency_list, H):
        self.adjacency_list = adjacency_list
        self.H=H

    #Trả về con/hàng xóm của v
    def get_neighbors(self, v):
        return adjacency_list[v]

    #hàm heuristic (h) chính là giá trị của node (node value)
    def h(self, n):
        return H[n]

    #Giải thuật A*
    def a_star_algroithm(self, start_node, end_node):
        #open_list: danh sách các node đã viếng thăm nhưng hàng xóm của nó chưa được viếng thăm
        #close_list: danh sách các node đã viếng thăm và hàng xóm của nó đã được viếng thăm

        open_list= set([start_node])
        closed_list= set([])

        #g chính là khoảng từ 1 node đến node khác
        #mảng g lưu trữ khoảng cách từ 1 node đến các node khác có liên kết với nó
        g={}
        g[start_node]=0

        #parents lưu trữ cha của node
        parents={}
        parents[start_node]=start_node

        while open_list: #open_list khác rỗng
            n=None

            #Tìm node có giá trị hàm f() nhỏ nhất
            # biết f=g+h
            for v in open_list:
                #Nếu giá trị f{v}<f{n}
                if n==None or (g[v]+self.h(v)<g[n]+self.h(n)):
                    n=v #gán n=v
```

```
        #Tìm node có giá trị hàm f() nhỏ nhất
        # biết f=g+h
        for v in open_list:
            #Nếu giá trị f{v}<f{n}
            if n==None or (g[v]+self.h(v)<g[n]+self.h(n)):
                n=v #gán n=v

        if n==None:
            print("Không tìm thấy đường đi")
            return None

        #Nếu node đang xét là end_node
        if n==end_node:
            #path: lưu trữ đường đi đến 1 node
            path=[]

            while parents[n]!=n:
                path.append(n)
                n=parents[n]

            #Thêm vào path nút start_node
            path.append(start_node)
            #Đảo mảng
            path.reverse()

            print("Đường đi: {}".format(path))
            return path

        #Ngược lại, vòng lặp cập nhật giá trị của các con/hàng xóm
        #Cập nhật giá trị h, g, f cho các node
        for(m, cost) in self.get_neighbors(n):
            #Nếu node đang xét chưa có trong open_list và close_list
            #thì thêm node vào open_list, cập nhật giá trị g và parent của node
            if m not in open_list and m not in closed_list:
                open_list.add(m)
                parents[m]=n
                g[m]=g[n]+cost

            #Ngược lại, nếu tìm được đường đi ngắn hơn
            # cập nhật giá trị, parent của node
            # nếu node đã duyệt rồi, lưu trong closed_list
            # nhưng nếu node này ở trong chuỗi đường đi ngắn hơn của node chưa xét
            #thì vẫn được rút ra và đưa vào open_list
            else:
                if g[m]>g[n]+cost:
                    g[m]=g[n]+cost
                    parents[m]=n

                    if m in closed_list:
                        closed_list.remove(m)
                        open_list.add(m)

        #Sau khi duyệt tất cả node hàng xóm/con của node n xong,
        #đưa node n vào closed_lisr
        open_list.remove(n)
        closed_list.add(n)

    print("Đường đi không tồn tại")
    return None
```

```
#-------------------------
#Khai báo đồ thị
adjacency_list={
        'A':[('C', 9),('D',7),('E',13),('F',20)],
        'C':[('H',6)],
        'D':[('E',4),('H',8)],
        'E':[('K',4),('I',3)],
        'F':[('I', 6), ('G',4)],
        'H':[('K',5)],
        'K':[('B', 6)],
        'I':[('K',9),('B',5)]}
#Khai báo giá trị hàm heuristic h()
H={
    'A':14,
    'B':0,
    'C':15,
    'D':6,
    'E':8,
    'F':7,
    'G':12,
    'H':10,
    'K':2,
    'I':4
    }
graph1=Graph(adjacency_list, H)
graph1.a_star_algroithm('A','B')
```

Bài 2. Cho bài toán sau, hãy tìm đường đi từ A đến B



A	366
B	0
C	160
D	242
F	178
G	77
L	244

M	241
O	380
P	98
R	193
S	253
T	329
Z	374

Khoảng cách ước lượng từ các đỉnh đến B

Yêu cầu:

a. Chạy tay giải thuật A* lưu vào file Word .

b. Áp dụng **giải thuật A*** để tìm đường đi từ A đến B cho bài toán sau