**Mục Lục**

TRƯỜNG ĐẠI HỌC GIAO THÔNG VÂN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------



**BÀI TẬP LỚN MÔN HỌC**

**THUẬT TOÁN VÀ ỨNG DỤNG**

**Đề tài: Thuật toán Bellman-Ford**

**Giảng viên hướng dẫn: Phạm Xuân Tích**

**Sinh viên thực hiện: Đặng Mạnh Hiếu**

**Mã sinh viên: 201210121**

**Lớp: Công nghệ thông tin 4 – K61**

**Hà Nội, Năm 2022**

**1. Nêu bài toán …………………………………………… 3**

**2. Nêu ý tưởng …………………………………………… 5**

**3. Mô phỏng tính toán bằng tay ………………………… 6**

**4. Code bằng Python …………………………………….. 8**

**5. Đánh giá thuật toán ………………………………….. 10**

**1. Nêu bài toán**

Bài toán tìm đường đi ngắn nhất trên đồ thị là một bài toán vô cùng đa dạng.

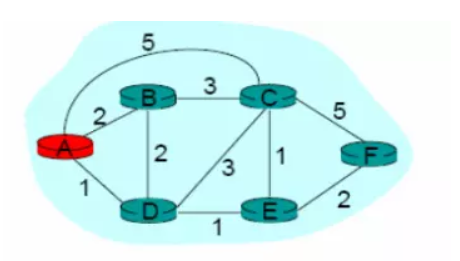
*Bài toán chung như sau:*

Cho một đồ thị gồm các đỉnh (các nodes) và khoảng cách giữa các đỉnh tương ứng. Tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh còn lại.

*Ví dụ:*

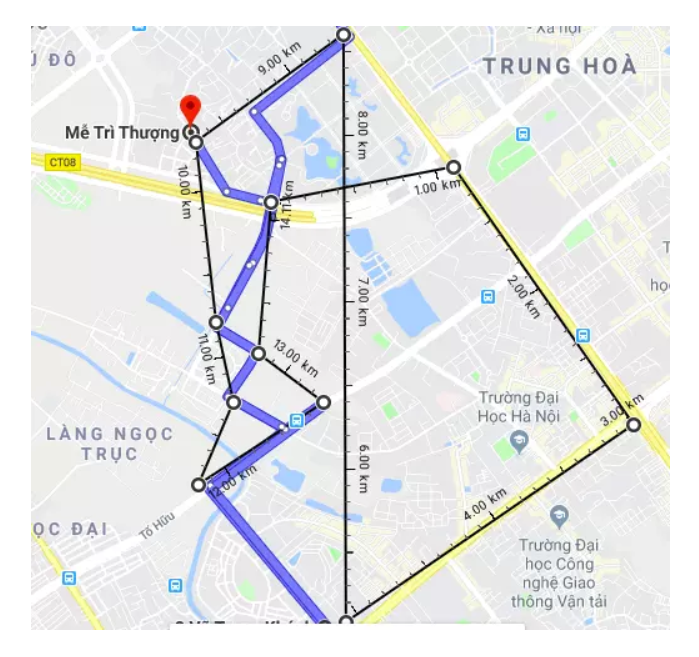
Cho một đồ thị gồm 6 đỉnh lần lượt là A, B, C, D, E, F với khoảng cách giữa các đỉnh tương ứng như dưới hình vẽ.

Tìm khoảng cách ngắn nhất từ đỉnh A tới các đỉnh còn lại.



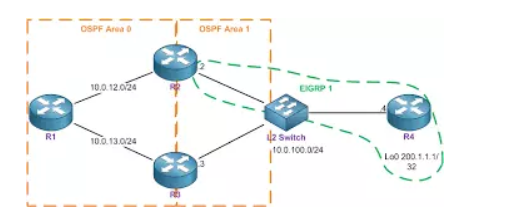
Để nói về ứng dụng của bài toán này, nếu chúng ta thay các đỉnh của đồ thị bằng cao giao lộ, các cạnh của nó sẽ là các tuyến đường, ta sẽ có một bài toán rất quen thuộc.

Bài toán tìm đường đi ngắn nhất đến một địa điểm trên bản đồ



Đây là một ví dụ rất thực tế với hiện nay, ứng dụng Google Map.

Ngoài ra, nếu ta thay các đỉnh bằng các router mạng hoặc các host, chúng ta sẽ có bài toán định tuyến đường đi của một hệ thống mạng.



Để giải quyết bài toán này, có khá nhiều thuật toán được đưa ra như: Dijiktra, Floyd-Warshall, Bellman-Ford, …

Mỗi thuật toán sẽ có ưu nhược điểm riêng tuy nhiên ở lần báo cáo này em sẽ sử dụng thuật toán Bellman-Fort để giải quyết bài toán.

**2. Nêu ý tưởng**

- Đầu tiên ta sẽ khởi tạo một list khoangcach để lưu lại khoảng cách từ đỉnh nguồn đến các đỉnh còn lại. Khi bắt đầu ta sẽ gán toàn bộ khoảng cách từ đỉnh nguồn đến các đỉnh khác bằng ∞ ngoại trừ khoảng cách từ đỉnh nguồn đến chính nó sẽ là 0.

- Ta thực hiện duyệt n lần, với n là số đỉnh của đồ thị.

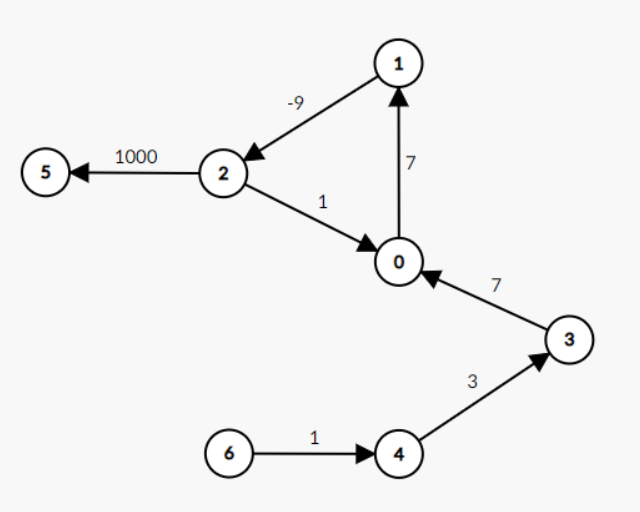
- Với mỗi lần duyệt, ta sẽ tiến hành kiểm tra khoảng cách từ đỉnh nguồn đến đỉnh cần xét.

Nếu khoảng cách từ đỉnh đang duyệt đến đỉnh cần xét mà khác ∞ và khoảng cách này + trọng số nhỏ hơn khoảng cách trước đó thì ta sẽ gán lại khoảng cách từ đỉnh đang duyệt đến đỉnh cần xét.

- Ta sẽ thực hiện duyệt n lần và nếu như ở lần duyệt thứ n mà còn bất kỳ cạnh nào có thể rút ngắn đường đi, điều đó chứng tỏ đồ thị có chu trình âm, ta kết thúc thuật toán.

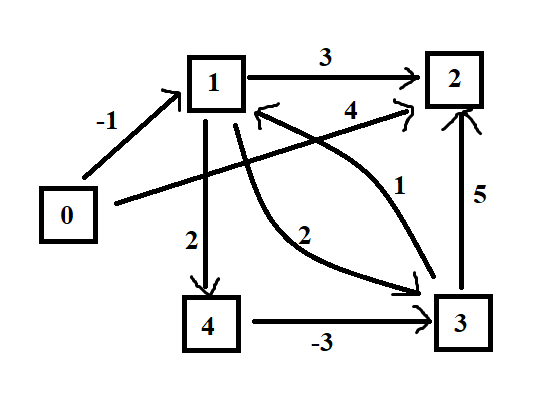
Chu trình âm là một chu trình trong đó tổng trọng số các cạnh là số âm.

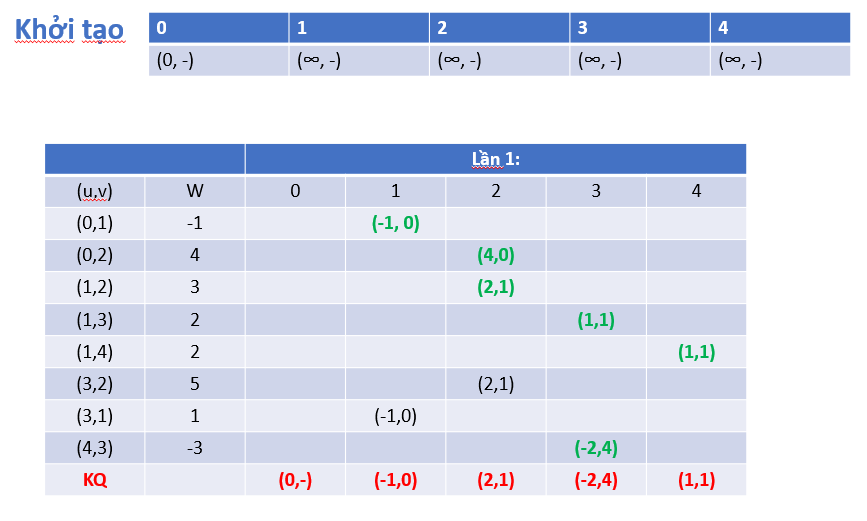
Ví dụ: Trong hình có một chu trình âm là: 0 → 1 → 2 có tổng trọng số là 7-9+1 = -1.

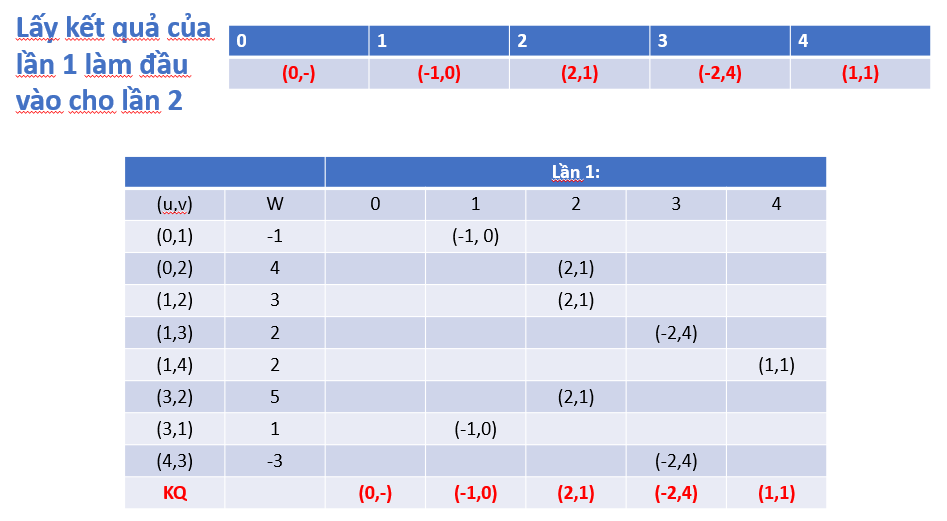


**3. Mô phỏng tính toán bằng tay**

Cho đồ thị như sau, tìm khoảng cách ngắn nhất từ đỉnh 0 đến các đỉnh còn lại.







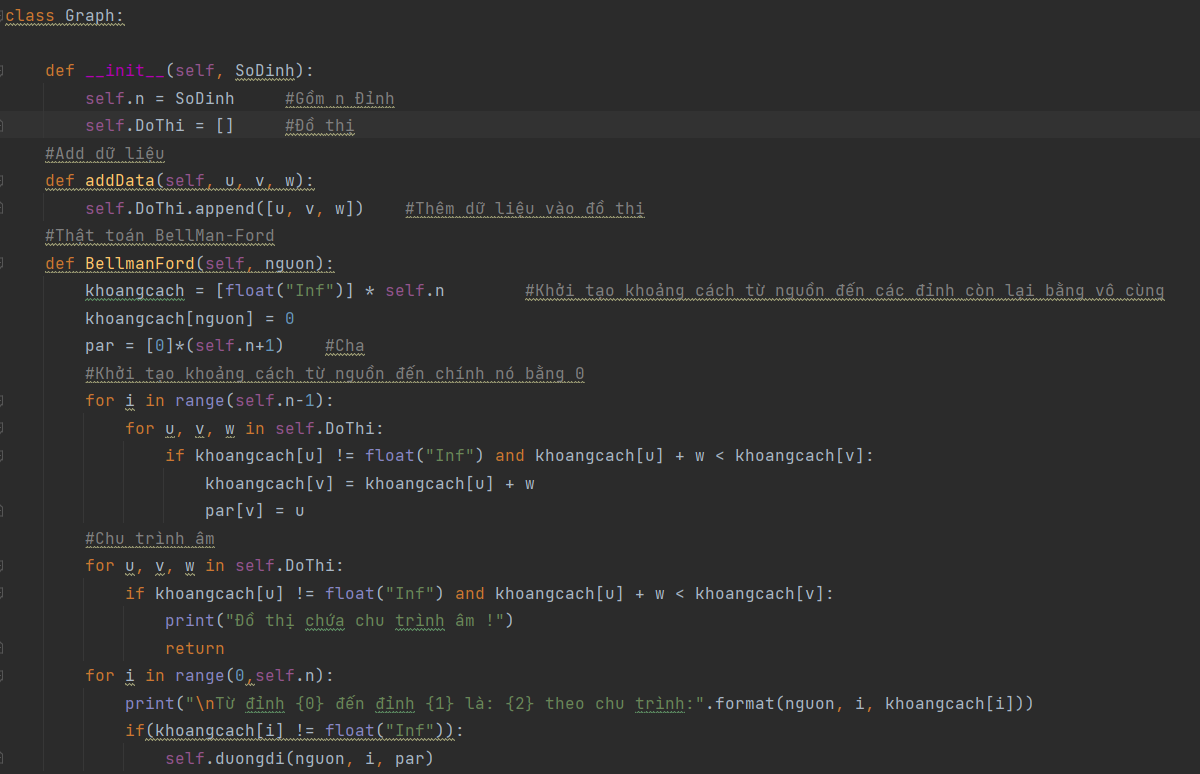
# Lần lặp thứ 2 có kết quả không khác lần lặp 1 → kết thúc thuật toán

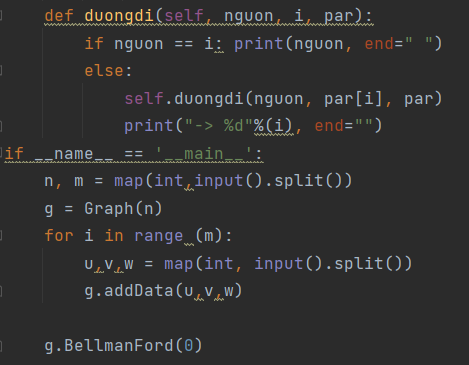
Vậy ta thu được kết quả là:

Từ đỉnh 0 đến đỉnh 0 có độ dài là 0  
Từ đỉnh 0 đến đỉnh 1 có độ dài là -1.

# Chu trình là: 0 → 1 Từ đỉnh 0 đến đỉnh 2 có độ dài là 2 Chu trình là: 0 → 1 → 2 Từ đỉnh 0 đến đỉnh 3 có độ dài là -2 Chu trình là: 0 → 1 → 4 → 3 Từ đỉnh 0 đến đỉnh 4 có độ dài là 1 Chu trình là: 0 → 1 → 4

# **4. Code Python**



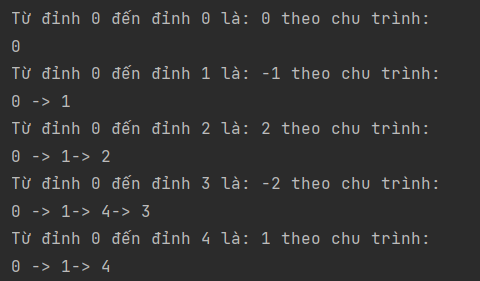


Test thử:

Input:

5 8  
0 1 -1  
0 2 4  
1 2 3  
1 3 2  
1 4 2  
3 2 5  
3 1 1  
4 3 -3

Output:



**5. Đánh giá thuật toán**

**Độ phức tạp về thời gian**

|  |  |
| --- | --- |
| Độ phức tạp của trường hợp tốt nhất | O (E) |
| Độ phức tạp của trường hợp trung bình | O(VE) |
| Độ phức tạp của trường hợp xấu nhất | O(VE) |

**Độ phức tạp về không gian**

Độ phức tạp về không gian của thuật toán là O(V)

**Trong đó:**V: số đỉnh của đồ thị  
E: số cung của đồ thị