

## Đường đi - Chu trình Euler

# Đường đi - Chu trình Euler

**Nguồn:** [Eulerian path - Wikipedia](#) [🔗](#) và một số nguồn khác

### Biên soạn:

- Bùi Nguyễn Ngọc Thắng - Carnegie Mellon University in Qatar

### Reviewer:

- Trần Quang Lộc - ITMO University
- Hoàng Xuân Nhật - VNUHCM-University of Science
- Nguyễn Nhật Minh Khôi - VNUHCM-University of Science
- Nguyễn Châu Khanh - VNU University of Engineering and Technology (VNU-UET)
- Hồ Ngọc Vĩnh Phát - VNUHCM-University of Science

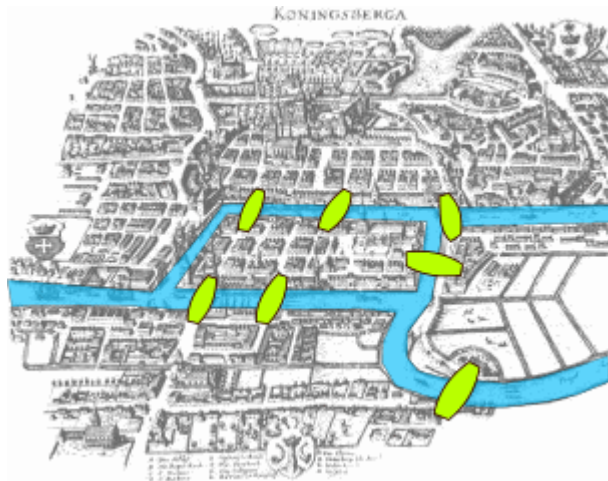
### Chuẩn bị bài tập và bộ test

- Nguyễn Hoàng Vũ - THPT chuyên Phan Bội Châu - Nghệ An

Bài viết sẽ giới thiệu cho độc giả về đường đi và chu trình Euler, một khái niệm cơ bản có ứng dụng rộng rãi trong lý thuyết đồ thị và lập trình thi đấu. Phạm vi bài viết bao gồm các định lý liên quan đến sự tồn tại của đường đi và chu trình Euler trong đồ thị và chứng minh, thuật toán tìm chu trình Euler và ứng dụng trong một số bài tập.

Khái niệm về đường đi và chu trình Euler lần đầu tiên được đề cập bởi Leonhard Euler năm 1736 khi nghiên cứu bài toán bảy cây cầu ở Königsberg.

Thành phố Königsberg thuộc Phổ (nay là Kaliningrad thuộc Nga), được chia làm 4 vùng bằng các nhánh sông Pregel. Các vùng này gồm 2 vùng bên bờ sông (B, C), đảo Kneiphof (A) và một miền nằm giữa hai nhánh sông Pregel (D). Vào thế kỷ XVIII, người ta đã xây 7 chiếc cầu nối những vùng này với nhau. Người dân ở đây tự hỏi: Liệu có cách nào xuất phát tại một địa điểm trong thành phố, đi qua 7 chiếc cầu, mỗi chiếc đúng 1 lần rồi quay trở về nơi xuất phát không?



Minh hoạ 7 cây cầu ở Königsberg - Nguồn: Wikipedia

Một cách tổng quát hơn:



Cho một đồ thị với tập các đỉnh và cạnh. Liệu có thể chỉ ra một đường đi hay chu trình đi qua tất cả các cạnh mỗi cạnh đúng một lần?

## Kí hiệu

Những kí hiệu sau sẽ được dùng xuyên suốt bài viết:

- ▶ Với đỉnh  $u$  thuộc một đồ thị **vô hướng** thì  $\deg(u)$  là bậc của  $u$ .
- ▶ Với đỉnh  $u$  thuộc một đồ thị **có hướng** thì  $\deg^+(u)$  là bán bậc ra của  $u$  và  $\deg^-(u)$  là bán bậc vào của  $u$ .

## Định nghĩa

- ▶ **Đường đi Euler** (Eulerian path/trail) trên một đồ thị (bất kể là vô hướng hay có hướng, đơn hay đa đồ thị) là đường đi qua tất cả các cạnh, mỗi cạnh đúng một lần.
- ▶ **Chu trình Euler** (Eulerian cycle/circuit/tour) trên một đồ thị là đường đi Euler trên đồ thị đó thoả mãn điều kiện đường đi bắt đầu và kết thúc tại cùng một đỉnh. Hiển nhiên rằng chu trình Euler cũng là một đường đi Euler.
- ▶ Đồ thị có chu trình Euler gọi là **đồ thị Euler** (Eulerian graph). Đồ thị chỉ có đường đi nhưng không có chu trình Euler được gọi là **đồ thị nửa Euler** (semi-Eulerian graph).

## Chu trình Euler trên đồ thị có hướng

### Định lý 1

Một đồ thị có hướng là đồ thị Euler nếu và chỉ nếu:

- Với mọi đỉnh  $u$  thuộc đồ thị,  $\deg^+(u) = \deg^-(u)$
- Tất cả đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông

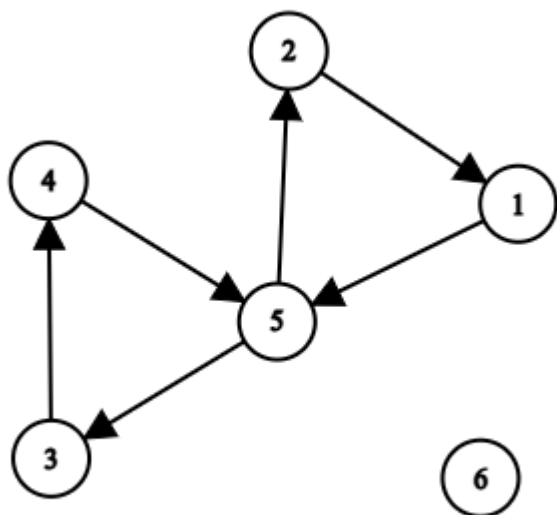
### Ví dụ

Đồ thị 1 (hình dưới) là một đồ thị Euler.

Ta thấy rằng mọi đỉnh  $u$  thuộc đồ thị đều có bán bậc ra bằng bán bậc vào:  $\deg^+(1) = 1 = \deg^-(1)$ ,  $\deg^+(5) = 2 = \deg^-(5)$ .

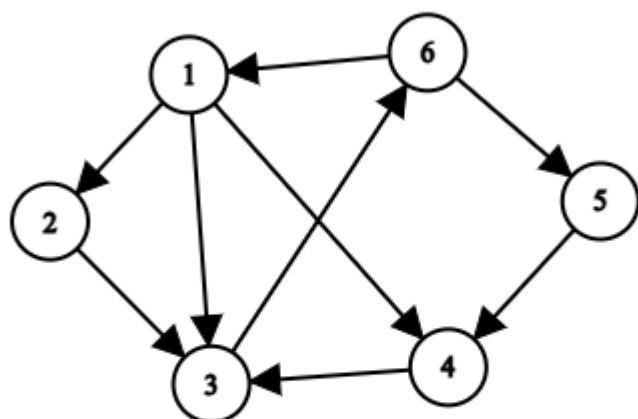
Đồng thời, các đỉnh có bậc lớn hơn 0 là 1, 2, 3, 4 và 5 đều nằm cùng một thành phần liên thông.

Do đó ta kết luận đồ thị 1 là đồ thị Euler. Thật vậy,  $2 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 2$  là một chu trình Euler trên đồ thị.



Đồ thị 1

Đồ thị 2 dưới có những đỉnh mà bán bậc vào khác bán bậc ra. Chẳng hạn, đỉnh 1 có 1 cạnh vào và 3 cạnh ra, tức  $\deg^+(1) = 1 \neq 3 = \deg^-(1)$ . Do đó ta kết luận đồ thị 2 không phải là đồ thị Euler.



Đồ thị 2

## Chứng minh

### Chiều thuận

Giả sử đồ thị có hướng tồn tại chu trình Euler.

- ▶ Vì chu trình Euler đi qua tất cả các cạnh nên cũng đi qua tất cả các đỉnh có bậc lớn hơn 0. Vì thế tất cả các đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông.
- ▶ Dễ thấy trong chu trình số lần ta đi vào một đỉnh bất kì bằng đúng số lần ta đi ra khỏi đỉnh đó nên với mỗi đỉnh  $u$ ,  $\deg^+(u) = \deg^-(u)$ .

### Chiều đảo

Ta sẽ sử dụng phương pháp quy nạp để chứng minh.

Vì các đỉnh có bậc bằng 0 không ảnh hưởng đến việc tồn tại chu trình Euler nên ta chỉ xét trường hợp đồ thị có hướng không tồn tại đỉnh có bậc là 0.

Nếu đồ thị có nhiều hơn một thành phần liên thông thì hiển nhiên đồ thị không tồn tại chu trình Euler.

Xét đồ thị có hướng  $G$  chỉ có một thành phần liên thông và tất cả các đỉnh trong đồ thị có bậc lớn hơn 0.

Ta cần chứng minh nếu với mỗi đỉnh  $u \in G$ ,  $\deg^+(u) = \deg^-(u)$  thì  $G$  tồn tại chu trình Euler.

Dễ thấy nếu  $G$  chỉ có 1 đỉnh thì định lý đúng.

### Bổ đề 1

Với mọi đỉnh  $u \in G$ , tồn tại một chu trình chứa  $u$ .

### Chứng minh

Ta dựng một chu trình bắt đầu từ  $u$ .

Bắt đầu từ  $u$  ta chọn một cạnh ra từ  $u$  đến  $u'$  chưa thăm bất kì để đi. Lặp lại thao tác này cho đến khi ta đến một đỉnh  $v$  mà ta đã thăm hết tất cả cạnh ra của  $v$ . Nếu  $v \neq u$  thì luôn tìm được một cạnh ra chưa thăm để đi tiếp vì khi đi vào thì ta giảm  $\deg^-(v)$  đi 1, tức ban đầu  $\deg^-(v) > 0$ . Do điều kiện mọi đỉnh thuộc  $G$  đều có bán bậc vào bằng bán bậc ra nên  $\deg^+(v) > 0$ , tức tồn tại cạnh ra chưa thăm. Suy ra  $v = u$ . Như vậy ta đã chỉ ra được một chu trình chứa  $u$  trong đồ thị.

Giả sử điều phải chứng minh đúng với mọi đồ thị con  $G' \subset G$  thoả điều kiện. Tức là ta tìm được một chu trình Euler  $C'$  trên mọi  $G' \subset G$  sao cho với mọi  $v \in G'$ ,  $\deg^-(v) = \deg^+(v)$ .

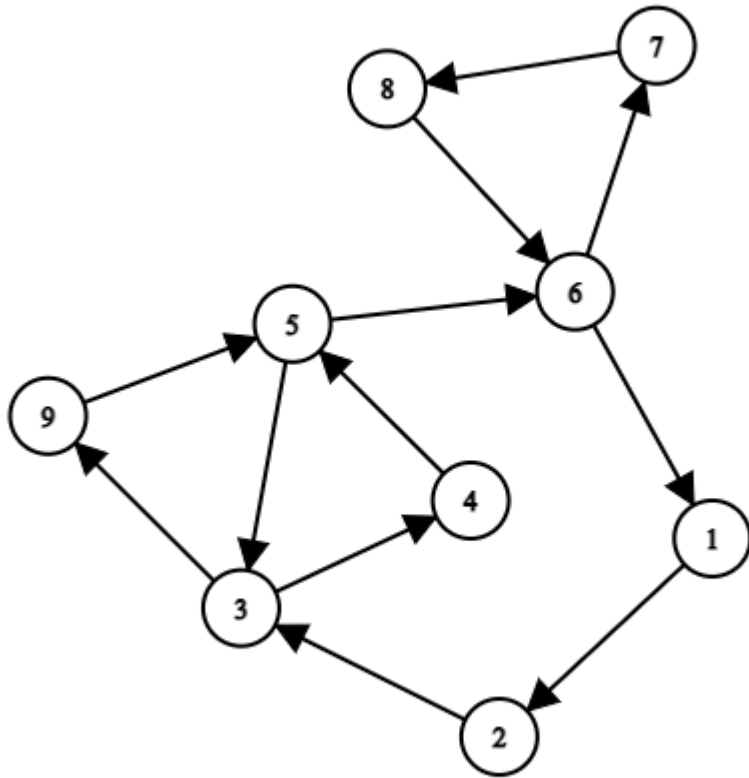
Ta chọn một đỉnh bất kì  $u \in G$ . Áp dụng bổ đề 1, giả sử ta tìm được một chu trình  $C$  chứa  $u$ . Xoá tất cả các cạnh trên  $G$  thuộc  $C$ .

Nếu sau khi xoá  $G$  không còn cạnh thì  $C$  là một chu trình Euler.

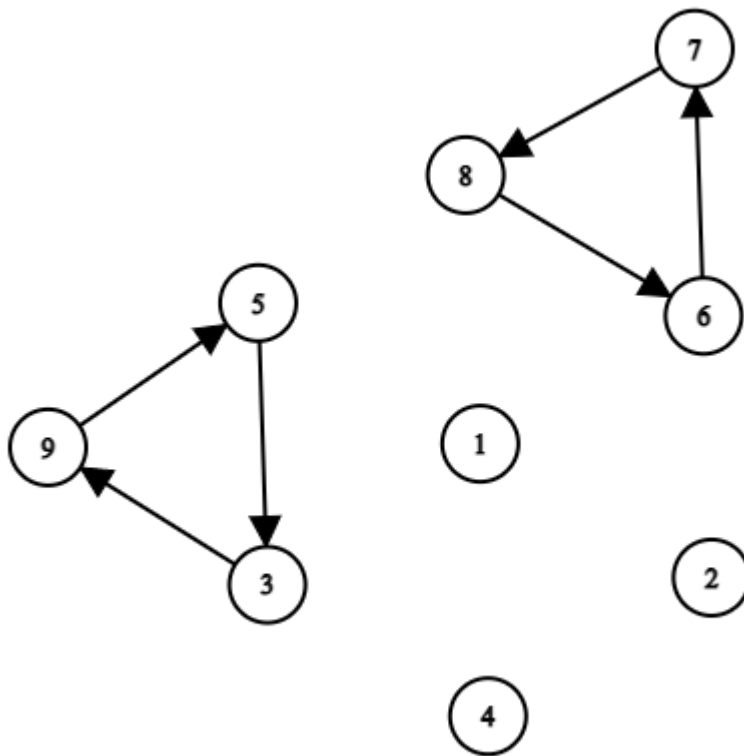
Ngược lại, ta thu được một số thành phần liên thông  $G_1, G_2, \dots, G_k$ . Do khi ta đi theo chu trình  $C$ , mỗi khi ta đi vào một đỉnh, ta đi ra khỏi đỉnh ấy ngay bước tiếp theo nên với mọi  $u \in G_i$  ( $1 \leq i \leq k$ ),  $\deg^-(u) = \deg^+(u)$ .

Theo giả sử thì với mọi  $G_i$  ta tìm được một chu trình Euler  $C_i$  đi qua tất cả các cạnh thuộc  $G_i$ . Nhận xét rằng mọi  $C_i$  có ít nhất một đỉnh chung với  $C$  do nếu  $C_i$  và  $C$  không có đỉnh chung nghĩa là  $C$  không đi qua bất kì đỉnh nào trong  $G_i$ , tức  $G$  và  $G_i$  không liên thông, trái với điều kiện đặt ra là mọi đỉnh thuộc  $G$  liên thông (lưu ý điều kiện đúng là mọi đỉnh có bậc lớn hơn 0 trong  $G$  thuộc cùng một thành phần liên thông, nhưng do ở đây ta đang xét trường hợp  $G$  liên thông yếu). Lần lượt nối các chu trình con  $C_i$  vào  $C$ , ta thu được một chu trình kết quả đi qua tất cả các cạnh trên  $G$ .

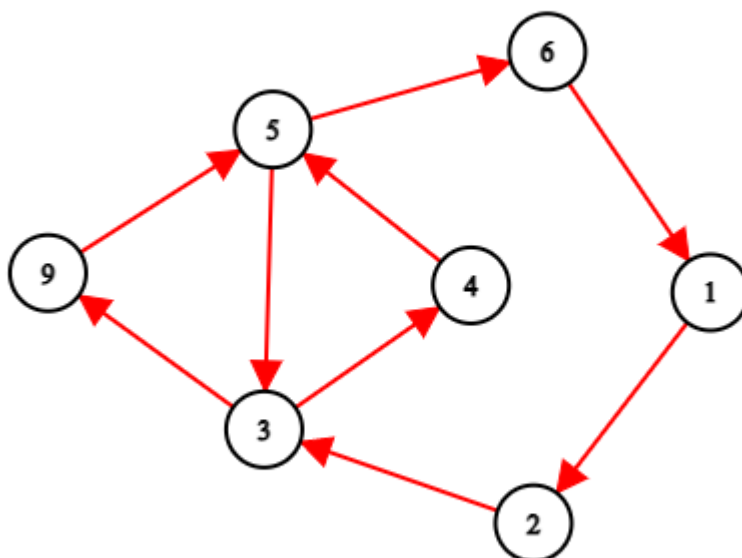
Đồ thị ban đầu, dễ thấy tồn tại chu trình  $C = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 1$



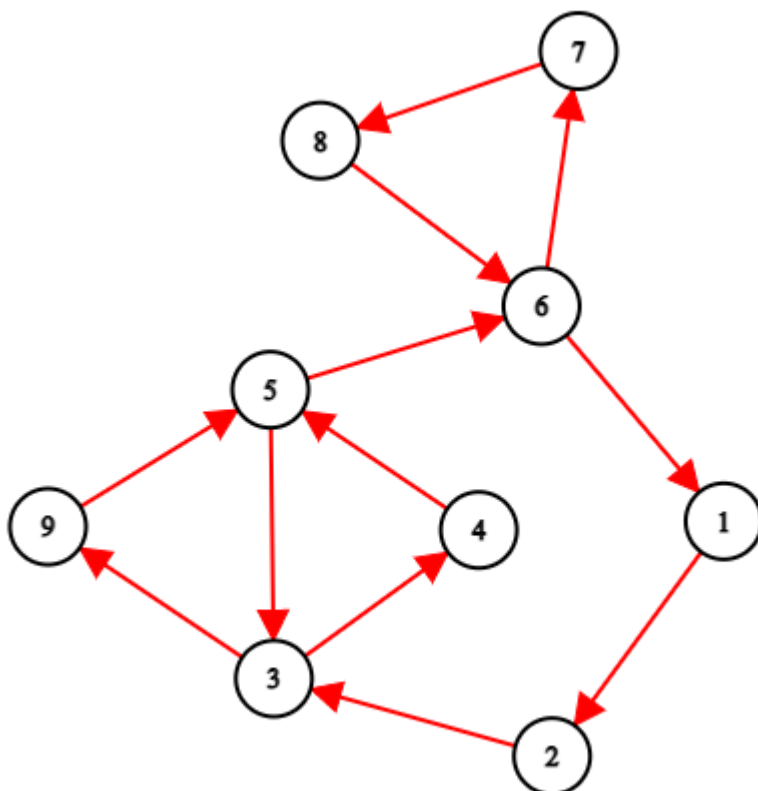
Xoá các cạnh thuộc  $C$ , ta nhận thấy có 2 chu trình con là  $C_1 = 3 \rightarrow 9 \rightarrow 5 \rightarrow 3$  và  $C_2 = 6 \rightarrow 8 \rightarrow 7 \rightarrow 6$



Nhận thấy giữa  $C$  và  $C_1$  có đỉnh chung 5. Nối 2 chu trình lại ta thu được một chu trình  $C$  mới lớn hơn:  $C = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 9 \rightarrow 5 \rightarrow 6 \rightarrow 1$ .



Tiếp tục, giữa  $C$  và  $C_2$  có đỉnh chung 6. Nối 2 chu trình lại ta thu được một chu trình  $C$  mới:  $C = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 9 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 6 \rightarrow 1$ . Ta đã đi qua tất cả các cạnh và tìm được chu trình Euler trong đồ thị.



## Đường đi Euler trên đồ thị có hướng

### Định lý 2

Đồ thị có hướng là đồ thị nửa Euler nếu và chỉ nếu đồ thị thoả **một trong hai** điều kiện sau:

1. Tồn tại chu trình Euler
2. Thoả tất cả các điều kiện sau:
  1. Tồn tại trong đồ thị đúng 2 đỉnh  $s$  và  $t$  sao cho  $\deg^-(s) = \deg^+(s) + 1$  và  $\deg^-(t) = \deg^+(t) - 1$
  2. Với mọi đỉnh  $u$  khác  $s$  và  $t$ ,  $\deg^+(u) = \deg^-(u)$
  3. Tất cả các đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông

Đỉnh  $s$  và đỉnh  $t$  cũng chính là đỉnh xuất phát và kết thúc của đường đi Euler trong đồ thị.

### Ví dụ

Đồ thị 3 dưới đây là một đồ thị nửa Euler.

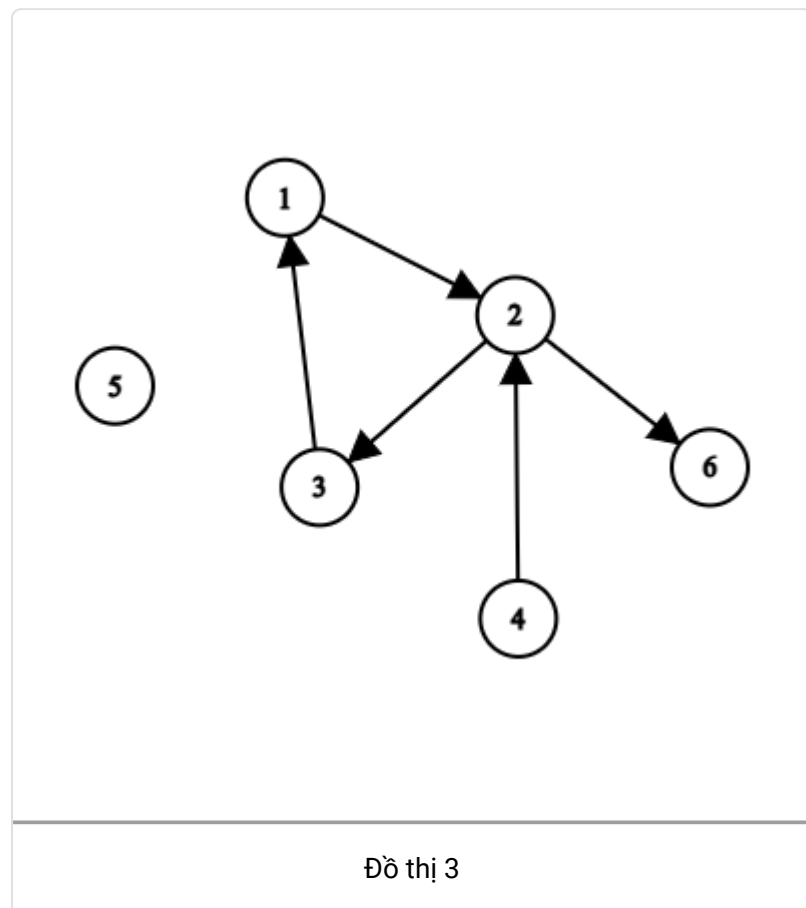
Đồ thị 3 có đỉnh 4 có  $\deg^-(4) = 0$  và  $\deg^+(4) = 1$  và đỉnh 6 có  $\deg^-(6) = 1$  và  $\deg^+(6) = 0$ .



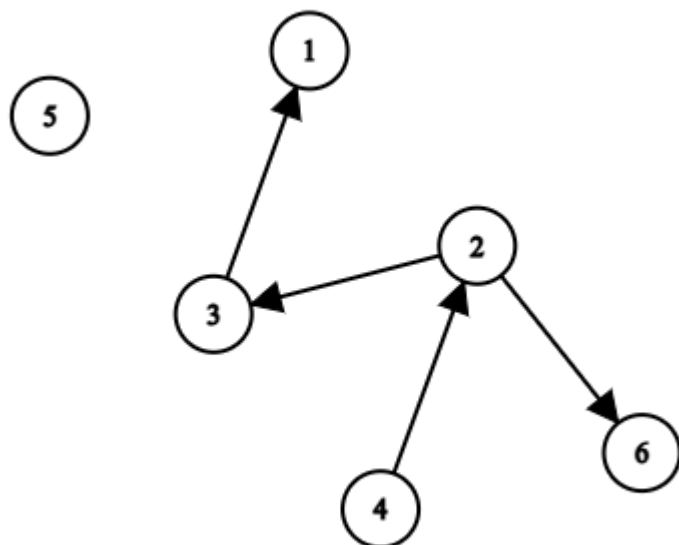
Các đỉnh khác ngoài 4 và 6 đều có bán bậc ra bằng bán bậc vào:  $\deg^-(1) = 1 = \deg^+(1)$ ,  $\deg^-(5) = 0 = \deg^+(5)$ , \ldots

Đồng thời các đỉnh có bậc lớn hơn 0 là 1, 2, 3, 4, 6 thuộc cùng một thành phần liên thông.

Do đó, ta kết luận tồn tại đường đi Euler trên đồ thị 3. Thật vậy,  $4 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 6$  là một đường đi Euler trên đồ thị.



Đồ thị 4 có hướng không tồn tại đường đi Euler do có các đỉnh có bậc ra khác bậc vào là 1, 2, 4, 6.



Đồ thị 4

## Chứng minh

### Chiều thuận

Giả sử tồn tại đường đi Euler bắt đầu từ  $s$  và kết thúc tại  $t$  trên đồ thị có hướng.

- Nếu  $s = t$ , đồ thị có chu trình Euler. Dựa vào định lý 1, ta chứng minh được định lý đúng trong trường hợp này.
- Nếu  $s \neq t$ , do đường đi Euler đi qua tất cả các cạnh nên nhận thấy:
  - $\deg^-(s) = \deg^+(s) + 1$  do đường đi xuất phát nhưng không kết thúc tại  $s$  nên  $\deg^-(s)$  phải ít hơn  $\deg^+(s)$  1 cạnh
  - $\deg^-(t) = \deg^+(t) - 1$  do đường đi kết thúc nhưng không xuất phát tại  $t$  nên  $\deg^-(t)$  phải nhiều hơn  $\deg^+(t)$  1 cạnh
  - $\deg^-(u) = \deg^+(u)$  với mọi  $u$  khác  $s$  và  $t$
  - Tất cả các đỉnh có bậc lớn hơn 0 trong đồ thị thuộc cùng một thành phần liên thông

### Chiều đảo

Trong trường hợp đồ thị có hướng tồn tại chu trình Euler, hiển nhiên đồ thị tồn tại đường đi Euler.

Giả sử đồ thị có hướng thoả các điều kiện trong định lý 2. Thêm một cạnh  $e$  nối từ  $t$  đến  $s$  vào đồ thị ( $s$  và  $t$  là 2 đỉnh duy nhất trong đồ thị không thoả điều kiện bán bậc vào bằng bán bậc ra). Khi này mọi đỉnh trong đồ thị đều có bán bậc vào bằng bán bậc ra nên đồ thị tồn tại chu trình Euler đi qua  $t$  và  $s$ :

$$u_1 u_2 u_3 \cdots t s \cdots u_{n-1} u_n$$

Ta nhận thấy trên đồ thị lúc sau tồn tại một đường đi qua tất cả các cạnh khác  $e$  bắt đầu từ  $s$  và kết thúc ở  $t$ .

$s \cdots u_{n-1} u_n u_1 u_2 u_3 \cdots t$

Do đó đồ thị ban đầu tồn tại một đường đi Euler.

## Chu trình Euler trên đồ thị vô hướng

### Định lý 3

Đồ thị vô hướng là đồ thị Euler nếu và chỉ nếu:

- Bậc của mọi đỉnh là chẵn
- Tất cả các đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông

### Chứng minh

Cách chứng minh định lý cho đồ thị vô hướng khá tương tự như cho đồ thị có hướng.

#### Chiều thuận

Giả sử đồ thị vô hướng tồn tại chu trình Euler.

- Vì chu trình Euler đi qua tất cả các cạnh nên dễ thấy tất cả các đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông.
- Vì chu trình Euler đi qua mỗi cạnh đúng một lần và với mỗi đỉnh, số lần ta đi vào bằng đúng số lần ta đi ra khỏi đỉnh ấy nên bậc của mỗi đỉnh là chẵn.

#### Chiều đảo

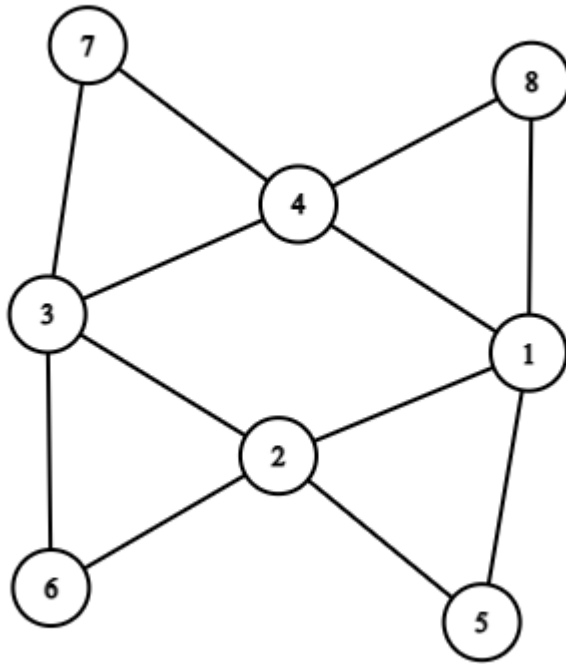
Ta cũng sử dụng phương pháp quy nạp để chứng minh.

Dễ thấy định lý đúng khi đồ thị có một đỉnh.

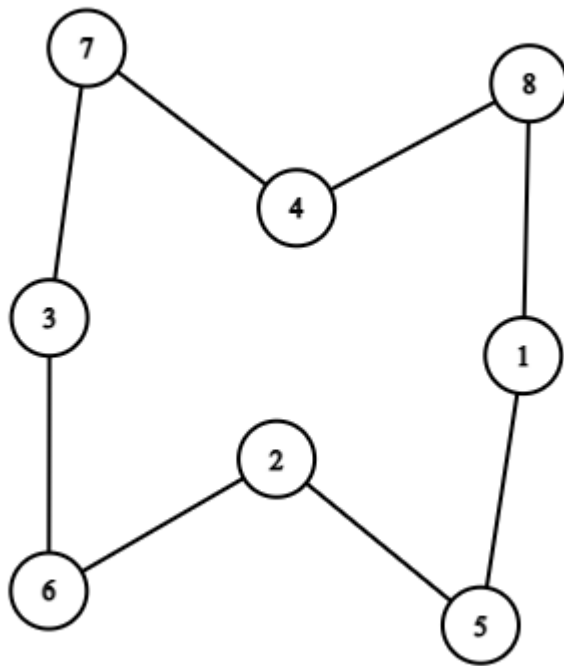
Xét một đồ thị vô hướng  $G$  có  $n$  đỉnh thoả  $\forall u \in G, 2 \mid \deg(u)$ . Tương tự như trên đồ thị có hướng, ta chứng minh được:

- Chọn một đỉnh  $u \in G$  bất kì thì luôn tìm được một chu trình  $C$  chứa  $u$ .
- Xoá các cạnh trên  $G$  thuộc  $C$ :
  - Nếu sau khi xoá  $G$  không còn cạnh thì  $C$  chính là một chu trình Euler.
  - Ngược lại, ta thu được một số thành phần liên thông  $G'$ .
    - Do trên chu trình  $C$ , mỗi khi ta đi vào một đỉnh ta lập tức đi ra khỏi đỉnh ấy nên sau khi xoá cạnh, bậc của mọi đỉnh vẫn chẵn.
    - Với mỗi  $G'$ , do bậc của mọi đỉnh vẫn chẵn, tìm được chu trình Euler  $C'$  đi qua tất cả các cạnh (giả thiết quy nạp).
    - Do tất cả các đỉnh có bậc lớn hơn 0 trong  $G$  thuộc cùng một thành phần liên thông nên mọi  $G'$  có đỉnh chung với  $C$ . Lần lượt nối các chu trình nhỏ  $C'$  vào  $C$ , ta thu được chu trình Euler của  $G$ .

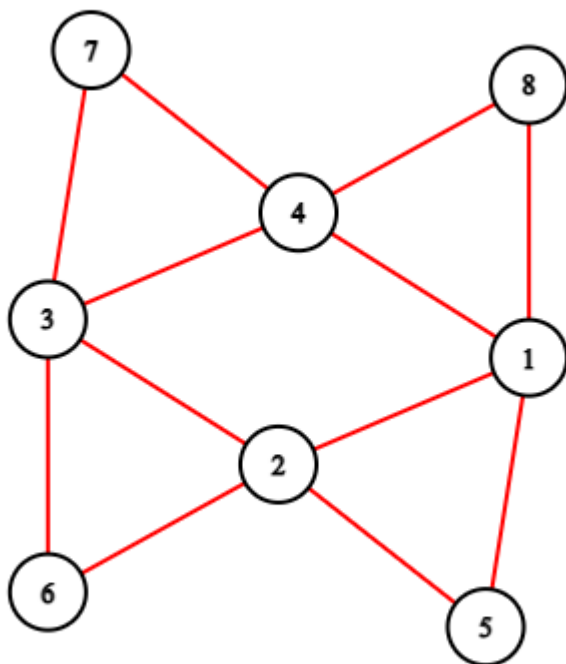
Đồ thị ban đầu, dễ thấy chu trình  $C = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$



Xoá các cạnh thuộc  $C$ , ta dễ thấy một chu trình  $C_1 = 7 \rightarrow 4 \rightarrow 8 \rightarrow 1 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 7$ .



Nhận thấy  $C$  và  $C_1$  có đỉnh chung 2, nối  $C$  và  $C_1$  để tạo ra chu trình mới lớn hơn:  
 $1 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 7 \rightarrow 4 \rightarrow 8 \rightarrow 1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ . Ta đã đi qua tất cả các cạnh và tìm được chu trình Euler trong đồ thị.



## Đường đi Euler trên đồ thị vô hướng

### Định lý 4

Tồn tại đường đi Euler trên một đồ thị vô hướng nếu và chỉ nếu:

- Đồ thị có **đúng** 0 hoặc 2 đỉnh bậc lẻ
- Tất cả các đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông

### Chứng minh

#### Chiều thuận

Giả sử đồ thị vô hướng tồn tại đường đi Euler xuất phát tại  $s$  và kết thúc tại  $t$ .

- Nếu  $s = t$ , dựa vào định lý 3, đồ thị có chu trình Euler nên tất cả đỉnh trong đồ thị có bậc chẵn và mọi đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông.
- Nếu  $s \neq t$ , do đường đi Euler đi qua tất cả các cạnh nên dễ thấy đồ thị có đúng 2 đỉnh bậc lẻ là  $s$  và  $t$  và các đỉnh có bậc lớn hơn 0 thuộc cùng một thành phần liên thông.

#### Chiều đảo

- Nếu tất cả các đỉnh trong đồ thị có bậc lớn hơn 0 thuộc cùng một thành phần liên thông và không có đỉnh bậc lẻ thì tồn tại chu trình Euler.

- Nếu tất cả các đỉnh trong đồ thị có bậc lớn hơn 0 thuộc cùng một thành phần liên thông và có đúng 2 đỉnh lẻ, ta thêm một cạnh ảo giữa 2 đỉnh lẻ. Khi này đồ thị tồn tại chu trình Euler. Xóa cạnh ảo khỏi chu trình Euler, ta thu được đường đi Euler của đồ thị.

## Thuật toán tìm chu trình - đường đi Euler

Qua phần chứng minh trên, ta có thể thấy nếu có thuật toán tìm chu trình Euler thì hoàn toàn có thể dễ dàng sử dụng để tìm đường đi Euler nên trong phần này chúng ta sẽ tập trung vào thuật toán tìm chu trình Euler.

Hai thuật toán tìm chu trình Euler được biết đến rộng rãi là thuật toán Fluery và thuật toán Hierholzer. Bài viết sẽ tập trung vào thuật toán Hierholzer do tính phổ biến và độ hiệu quả của thuật toán này so với thuật toán Fluery.

### Thuật toán Hierholzer cho đồ thị có hướng

Thuật toán Hierholzer tìm chu trình Euler dựa trên các bước đã nêu trong phần chứng minh định lý cho đồ thị có hướng.

Giả sử đồ thị thỏa định lý 1. Các bước trong thuật toán cụ thể như sau:

0. Trước khi bắt đầu thuật toán ta chọn một đỉnh  $u$  bất kì có bậc lớn hơn 0 trong đồ thị để bắt đầu. Ta xem  $u$  là tham số cho thuật toán. Nếu đồ thị không có cạnh, hiển nhiên ta trả về chu trình rỗng.
1. Từ đỉnh  $u$  tham số, ta tìm một chu trình  $C$  chứa  $u$ . Các bước cụ thể để tìm  $C$  như sau:
  1. Khởi tạo mảng kết quả  $C$  là rỗng và xuất phát từ đỉnh  $u$ .
  2. Từ đỉnh đang đứng, chọn một cạnh ra chưa thăm để đi. Đánh dấu cạnh vừa đi qua và cập nhật vào  $C$ .
  3. Lặp lại bước (ii). Nếu không thể đi tiếp thì ta tìm được một chu trình  $C$  và sang bước 2.
2. Nếu ta đã đi qua tất cả cạnh, trả  $C$  là kết quả. Nếu  $C$  chưa đi qua tất cả các cạnh, các cạnh chưa thăm trong đồ thị tạo thành những thành phần liên thông yếu. Tìm một đỉnh  $v$  trên đồ thị thuộc  $C$  mà có cạnh ra chưa đi qua.
3. Gọi đệ quy thủ tục tìm chu trình Euler với tham số là đỉnh  $v$ . Sau khi hoàn thành ta thu được một chu trình Euler  $D$  đi qua tất cả các cạnh trong đồ thị con chứa  $v$ .
4. Nối hai chu trình  $C$  và  $D$  tại đỉnh chung  $v$ . Lặp lại bước 2.

### Cài đặt mẫu

Trong phần cài đặt mẫu, đồ thị có hướng được biểu diễn dưới dạng danh sách kề. Trong trường hợp đồ thị vô hướng, cài đặt tương tự nhưng khi đánh dấu cạnh đã thăm lưu ý đánh dấu cả hai chiều.

Ngoài ra cài đặt mẫu sử dụng cấu trúc danh sách liên kết đôi thông qua cấu trúc `list` trong thư viện chuẩn của C++ để biểu diễn chu trình  $C$ . Việc sử dụng danh sách liên kết đôi giúp bước nối hai chu trình trở nên đơn giản hơn do ta chỉ cần thay đổi liên kết giữa các phần tử trong danh sách. Ngoài cách sử dụng danh sách liên kết đôi, một số cài đặt khác sử dụng hai ngăn xếp để nối hai chu trình. Độc giả có thể tham khảo cách cài đặt này ở các tài liệu khác.

```
1 | // N - số đỉnh nhiều nhất
2 | // M - số cạnh nhiều nhất
```

```

3
4 struct Edge {
5     int target, id;
6
7     Edge(int _target, int _id): target(_target), id(_id) {}
8 };
9
10 vector<Edge> adj[N]; // Danh sách kề lưu cạnh và chỉ số
11 bool used_edge[M]; // Mảng đánh dấu cạnh đã thăm
12
13 list<int> euler_walk(int u) {
14     // Sử dụng cấu trúc danh sách liên kết để lưu kết quả
15     list<int> ans;
16
17     // Xuất phát từ đỉnh u
18     ans.push_back(u);
19
20     while (!adj[u].empty()) {
21         // Chọn một cạnh bất kì chưa thăm
22         int v = adj[u].back().target;
23         int eid = adj[u].back().id;
24
25         // Xoá cạnh vừa đi qua khỏi đồ thị
26         // Lưu ý việc xoá cạnh có thể **ảnh hưởng** tới các
27         // thao tác trên đồ thị về sau do việc xoá cạnh sẽ
28         // **phá huỷ** hoàn toàn danh sách cạnh
29         // Nên sao lưu danh sách cạnh ra biến khác nếu cần dùng lại
30         adj[u].pop_back();
31
32         // Bỏ qua nếu cạnh đã thăm
33         if (used_edge[eid]) continue;
34
35         // Đánh dấu cạnh đã đi qua
36         used_edge[eid] = true;
37
38         // Di chuyển sang đỉnh mới
39         u = v;
40
41         // Thêm cạnh vào đường đi hiện tại
42         // Có nhiều cách lưu chu trình như lưu đỉnh, cạnh,
43         // chỉ số cạnh, ...
44         ans.push_back(u);
45     }
46
47     // Tìm cạnh chưa thăm từ một đỉnh trên chu trình hiện tại
48     // Bắt đầu từ đỉnh thứ hai trong chu trình do ta biết
49     // rằng đỉnh đầu tiên trong chu trình (u) đã không còn
50     // cạnh ra
51     for (auto it = ++ans.begin(); it != ans.end(); ++it) {
52         // Gọi đệ quy tiếp tục tìm chu trình mới
53         auto t = euler_walk(*it);
54     }
55 }

```



```

55
56         // Nối chu trình tìm được vào chu trình hiện tại
57         t.pop_back();
58         ans.splice(it, t);
59     }
60
61     return ans;
}

```

Độ phức tạp thời gian của cách cài đặt trên là  $O(m)$  do việc nối hai chu trình được thực hiện trong thời gian  $O(1)$  nếu chu trình được biểu diễn bằng danh sách liên kết. Nếu chúng ta biểu diễn chu trình bằng một số cấu trúc phổ biến khác như (chẳng hạn `vector`) thì độ phức tạp thời gian sẽ tăng lên do việc nối chu trình kém hiệu quả.

Độ phức tạp bộ nhớ là tuyến tính dựa vào số đỉnh và số cạnh.

## Ứng dụng

### CSES - Teleporters Path

#### Tóm tắt đề

Tìm một đường đi Euler trên đồ thị có hướng  $N \leq 10^5$  đỉnh,  $M \leq 2 \cdot 10^5$  cạnh. In **IMPOSSIBLE** nếu không thể tìm được.

#### Lời giải

Như đã đề cập, để tìm đường đi Euler, ta thêm một cạnh ảo từ giữa 2 đỉnh lẻ, tìm chu trình Euler, rồi xoá cạnh ảo đã thêm.

Một cách khác để tìm đường đi Euler là ta chỉ cần gọi thủ tục tìm chu trình Euler như trên với tham số là đỉnh 1. Kết quả nhận được là đường đi Euler trên đồ thị. Lý giải là khi gọi thủ tục tìm chu trình Euler trong trường hợp này, ở lần lặp đầu tiên, chúng ta sẽ tìm được một đường đi từ 1 tới  $n$ . Những cạnh chưa thăm còn lại trong đồ thị tạo thành những thành phần liên thông thoả điều kiện tồn tại chu trình Euler. Khi này thuật toán sẽ gọi đệ quy tìm chu trình Euler trên từng đồ thị con và nối các chu trình con lại để dựng đường đi cần tìm.

#### Cài đặt

```

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 2, M = 2e5 + 2;

struct Edge {
    int target, id;

    Edge(int _target, int _id): target(_target), id(_id) {}
};

```

```
int n, m, in_deg[N];
vector<Edge> adj[N];
bool used_edge[M];

list<int> euler_walk(int u); // Hàm euler_walk như cài đặt mẫu

bool check() {
    if (adj[1].size() != in_deg[1] + 1 ||
        adj[n].size() != in_deg[n] - 1)
        return false;

    for (int i = 2; i < n; ++i)
        if (adj[i].size() != in_deg[i])
            return false;

    return true;
}

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u].emplace_back(v, i);
        ++in_deg[v];
    }

    // Kiểm tra tồn tại đường đi Euler
    if (!check()) {
        cout << "IMPOSSIBLE";
        return 0;
    }

    // Thêm cạnh ảo vào đồ thị
    adj[n].emplace_back(1, m);

    // Tìm chu trình Euler
    list<int> ans = euler_walk(1);

    // Kiểm tra xem đã đi qua hết tất cả cạnh chưa vì trong trường hợp đồ
    // thị không liên thông ta không thể đi qua tất cả cạnh
    if (ans.size() < m + 1)
        cout << "IMPOSSIBLE";
    else {
        // Tìm cạnh ảo đã thêm
        for (auto u1 = ans.begin(), u2 = ++ans.begin(); u2 != ans.end(); ++u1,
            if (*u1 == n && *u2 == 1) {
                // In các cạnh trên chu trình nhưng bỏ qua cạnh ảo
                for (auto i = u2; i != ans.end(); ++i)
                    cout << (*i) << " ";
                for (auto i = ++ans.begin(); i != u2; ++i)
```

```

64 |
65 |         cout << (*i) << " ";
66 |         break;
67 |     }
68 | }
69 | return 0;
    | }

```

## VNOI Marathon 08 - Mê cung

### Tóm tắt đề

Cho đơn đồ thị vô hướng  $N \leq 200$  đỉnh,  $M$  cạnh. Mỗi cạnh  $(u, v)$  có trọng số là  $W_{u,v} \leq 10000$ . Tìm một chu trình đi qua tất cả các cạnh của đồ thị, sao cho tại mỗi thời điểm, tổng trọng số các cạnh đi qua không âm.

### Lời giải

Nếu đồ thị không liên thông, có đỉnh bậc lẻ, hay tổng trọng số tất cả các cạnh âm thì hiển nhiên không thể tìm được chu trình thoả yêu cầu.

Tìm một chu trình Euler bất kì trên đồ thị  $C = u_0 u_1 \dots u_m$ . Lưu ý do  $C$  là một chu trình nên đỉnh liền sau  $u_m$  là  $u_0$ .

Gọi  $S$  là mảng tổng tiền tố của trọng số các cạnh trên  $C$ . Như vậy,

$$S_i = \sum_{j=0}^i W_{u_j, u_{j+1}}$$

Lưu ý  $u_{j+1}$  chỉ đỉnh liền sau  $u_j$  trong  $C$ .

Ta biết rằng  $S_m \geq 0$  do như đã đề cập, nếu tổng trọng số các cạnh âm thì không tìm được chu trình thoả yêu cầu.

Gọi  $u_k$  là đỉnh thuộc chu trình sao cho  $S_k$  nhỏ nhất.

Nếu  $S_k \geq 0$  thì  $C$  thoả yêu cầu.

Nếu  $S_k < 0$  thì  $k < m$  do  $S_m \geq 0$ .

Như vậy với mọi  $k < k_1 \leq m$  thì  $S_{k_1} - S_k \geq 0$  hay tổng trọng số các cạnh giữa vị trí  $k$  và  $k_1$  trong  $C$  không âm. Tương tự với mọi  $0 \leq k_2 < k$  thì  $S_{k_2} - S_k \geq 0$ .

Nhận thấy khi này nếu ta dịch chu trình  $C$  ban đầu sao cho  $u_k$  là đỉnh xuất phát thì sẽ tìm được một chu trình  $C' = u_k \dots u_m u_0 u_1 \dots u_{k-1}$  thoả yêu cầu do:

- Với mọi  $k < k_1 \leq m$ ,  $W_{u_k, u_{k+1}} + \dots + W_{u_{k_1-1}, u_{k_1}} = S_{k_1} - S_k \geq 0$ .
- Với mọi  $0 \leq k_2 < k$ ,  $W_{u_k, u_{k+1}} + \dots + W_{u_{m-1}, u_m} + W_{u_0, u_1} + \dots + W_{u_{k_2-1}, u_{k_2}} = (S_m - S_k) + S_{k_2} = S_m + (S_{k_2} - S_k) \geq 0$

### Cài đặt

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 202;
6  const int M = 40002;
7
8  struct Edge {
9      int target, id;
10
11      Edge(int _target, int _id): target(_target), id(_id) {}
12 };
13
14 int n, m, w[M], deg[N], edge_id[N][N], S[M];
15 vector<Edge> adj[N];
16 bool used_edge[M];
17
18 list<int> euler_walk(int u); // Hàm euler_walk như cài đặt mẫu
19
20 int main() {
21     cin >> n >> m;
22     for (int i = 0; i < m; ++i) {
23         int u, v;
24         cin >> u >> v >> w[i];
25
26         // Thêm cạnh vào đồ thị
27         adj[u].emplace_back(v, i);
28         adj[v].emplace_back(u, i);
29
30         edge_id[u][v] = i;
31         edge_id[v][u] = i;
32
33         // Cập nhật bậc của đỉnh
34         ++deg[u];
35         ++deg[v];
36     }
37
38     // Nếu đồ thị có đỉnh bậc lẻ hiển nhiên không tồn tại đáp án
39     for (int i = 1; i <= n; ++i)
40         if (deg[i] % 2 != 0) {
41             cout << -1;
42             return 0;
43         }
44
45     // Tìm chu trình Euler
46     list<int> cycle = euler_walk(1);
47     int cycle_size = cycle.size();
48
49     // Nếu các đỉnh có bậc lớn hơn 0 trong đồ thị không nằm cùng một TPLT
50     // thì không tồn tại đáp án
51

```

```

51
52     if (cycle_size < m + 1) {
53         cout << -1;
54         return 0;
55     }
56
57     // Tạo mảng S như đã đề cập
58     fill(S, S + M, 0);
59     int u = *cycle.begin();
60     auto it = ++cycle.begin();
61     for (int i = 1; it != cycle.end(); ++i, ++it) {
62         int v = *it;
63         S[i] = S[i - 1] + w[edge_id[u][v]];
64         u = v;
65     }
66
67     // Tìm k sao cho S[k] -> min
68     int k = 0;
69     for (int i = 1; i < cycle_size; ++i)
70         if (S[i] < S[k]) k = i;
71
72     // Dịch chu trình sao cho bắt đầu ở u[k]
73     list<int> ans;
74     ans.clear();
75     it = cycle.begin();
76     for (int i = 0; i < k; ++i, ++it)
77         ans.push_back(*it);
78     ans.push_back(*it);
79     ans.insert(ans.begin(), it, (--cycle.end()));
80
81     for (int i : ans) cout << i << " ";
82
83     return 0;
}

```

## 10040 - Ouroboros Snake


### Tóm tắt đề

Số Ouroboros "bậc"  $n$  là số nhị phân được định nghĩa như sau:

- Đặt  $2^n$  bit nhị phân thành một vòng tròn. Nếu ta có thể thu được  $2^n$  dãy nhị phân độ dài  $n$  khác nhau thì số đó là số Ouroboros.

Hãy trả lời các truy vấn với hai tham số  $n$  và  $k$ . Với mỗi truy vấn, cho biết bit thứ  $k$  (đánh số bắt đầu từ 0) trong số Ouroboros độ dài  $n$  nhỏ nhất theo thứ tự từ điển ( $0 < n < 22, 0 \leq k < 2^n$ ).

### Lời giải

Bài toán này đề cập đến khái niệm về dãy de Bruijn. Độc giả có thể tìm hiểu kỹ hơn tại đây [de Bruijn sequence - Wikipedia](#) . Bài viết chỉ đề cập đến những ý cần biết để giải quyết bài toán.

Dãy de Bruijn  $B(k, n)$  bậc  $n$  của một tập  $S$  gồm  $k$  phần tử là một dãy vòng tròn sao cho với mỗi dãy  $T$  độ dài  $n$  chỉ gồm các phần tử thuộc  $S$ ,  $T$  xuất hiện đúng một lần trong  $B(k, n)$  dưới dạng một dãy con liên tiếp. Hiển nhiên  $B(k, n)$  có  $k^n$  phần tử.

Đồ thị de Bruijn là một khái niệm liên quan mật thiết đến dãy de Bruijn. Đồ thị de Bruijn  $n$  chiều của một tập  $S$  có  $k$  phần tử là đồ thị có  $k^n$  đỉnh, mỗi đỉnh  $u$  đại diện cho một dãy  $T_u$  độ dài  $n$  chỉ gồm các phần tử trong  $S$ . Giữa hai đỉnh  $u$  và  $v$  có cạnh một chiều từ  $u$  đến  $v$  nếu **hậu tố** độ dài  $n - 1$  của  $T_u$  trùng với **tiền tố** độ dài  $n - 1$  của  $T_v$ .

Một tính chất quan trọng cần lưu ý là với cùng một tập  $S$ , đồ thị de Bruijn  $n$  chiều là đồ thị đường (*line graph*) của đồ thị de Bruijn  $n - 1$  chiều.

Đồ thị đường  $L(G)$  của một đồ thị vô hướng  $G$  là đồ thị được xây dựng như sau:

- ▶ Với mỗi cạnh trong  $G$ , tạo một đỉnh tương ứng trên  $L(G)$
- ▶ Hai đỉnh bất kì trên  $L(G)$  được nối bởi một cạnh vô hướng nếu hai cạnh tương ứng với hai đỉnh đó có đỉnh chung trên  $G$

Nhận thấy rằng dãy de Bruijn  $B(k, n)$  chính là một chu trình Hamilton của đồ thị de Bruijn  $n$  chiều và  $k$  phần tử. Một hệ quả đáng lưu ý của các tính chất trên là do chu trình Hamilton của một đồ thị  $G$  là chu trình Euler của đồ thị đường của  $G$ , nên  $B(k, n)$  cũng chính là chu trình Euler của đồ thị de Bruijn  $n - 1$  chiều và  $k$  phần tử.

Trong bài toán này, với mỗi giá trị  $0 < n < 22$ , ta có thể tìm trước dãy de Bruijn  $B(n, 2)$  thứ tự từ điển nhỏ nhất để có thể truy cứu lại với mỗi truy vấn. Ta tận dụng hệ quả nêu trên để tìm số Ouroboros bậc  $n$  thứ tự từ điển nhỏ nhất. Trước tiên, ta dựng đồ thị de Bruijn  $n - 1$  chiều với 2 phần tử 0 và 1: biểu diễn các số nhị phân từ 0 đến  $2^n - 1$  dưới dạng một đồ thị có  $2^n$  đỉnh, đỉnh  $i$  thể hiện số  $i$  dưới dạng nhị phân. Giữa 2 đỉnh  $u$  và  $v$  của đồ thị có cạnh một chiều từ  $u$  đến  $v$  nếu  $u = b_1b_2b_3 \dots b_n$  và  $v = b_2b_3 \dots b_nb_{n+1}$ . Ta đặt trọng số của cạnh  $(u, v)$  bằng với  $b_{n+1}$ .

Như vậy ban đầu mỗi đỉnh trong đồ thị có 2 cạnh ra - 1 cạnh có trọng số là 0 và 1 cạnh có trọng số là 1 - và 2 cạnh vào cũng có trọng số là 0 và 1. Do mỗi đỉnh trong đồ thị có bán bậc ra bằng bán bậc vào nên đồ thị vừa dựng là một đồ thị Euler. Khi này bài toán quy về tìm chu trình Euler có thứ tự từ điển nhỏ nhất trên đồ thị.

Ta dùng phương pháp tham lam để giải quyết yêu cầu tìm thứ tự từ điển nhỏ nhất:

- ▶ Khi tìm chu trình xuất phát từ một đỉnh, ta sẽ ưu tiên đi qua cạnh có trọng số là 0 trước.
- ▶ Sau khi có một chu trình bắt đầu từ một đỉnh, các cạnh còn lại trong đồ thị tạo thành các thành phần liên thông. Theo thuật toán, khi này ta tìm một đỉnh trên chu trình hiện tại có cạnh chưa thăm và gọi đệ quy tiếp tục tìm chu trình bắt đầu từ đỉnh đó. Do mỗi đỉnh chỉ có 2 cạnh nên khi này nếu đỉnh còn cạnh chưa thăm, cạnh đó chắc chắn có trọng số 1. Do ta cần thứ tự từ điển nhỏ nhất nên ta muốn đặt các cạnh trọng số 1 càng về cuối chu trình càng tốt. Vì thế ta sẽ duyệt chu trình hiện tại từ cuối về đầu, và gọi đệ quy thủ tục nếu có đỉnh có cạnh chưa thăm.

Với mỗi bậc  $n$ , độ phức tạp thời gian sẽ là  $O(2^n)$ . Như vậy độ phức tạp thời gian để tiền xử lí cho toàn bộ bậc sẽ là khoảng  $1 + 2 + \dots + 2^{21} \approx 2^{22}$  phép tính, đủ trong giới hạn cho phép.

## Cài đặt

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  struct Edge {
6      int target, weight;
7
8      Edge(int _target, int _weight): target(_target), weight(_weight) {}
9  };
10
11 vector<vector<Edge>> build_graph(int n) {
12     vector<vector<Edge>> g(1 << n);
13     for (int u = 0; u < (1 << n); ++u) {
14         // Lưu ý thêm cạnh có trọng số 1 trước để ưu tiên sử dụng cạnh
15         // có trọng số 0 khi tìm chu trình
16         g[u].emplace_back((u << 1) + 1 & ((1 << n) - 1), 1);
17         g[u].emplace_back((u << 1) & ((1 << n) - 1), 0);
18     }
19     return g;
20 }
21
22 list<Edge> euler_walk(int u, vector<vector<Edge>> &g) {
23     list<Edge> ans;
24
25     while (!g[u].empty()) {
26         int v = g[u].back().target;
27         ans.push_back(g[u].back());
28         g[u].pop_back();
29         u = v;
30     }
31
32     // Duyệt chu trình hiện tại từ cuối về đầu và
33     // gọi đệ quy nếu cần thiết
34     auto it = --ans.end();
35     while (it != ans.begin())
36     {
37         auto tit = it;
38         --tit;
39         ans.splice(it, euler_walk(tit->target, g));
40         it = tit;
41     }
42
43     return ans;
44 }
45
46 const int N = 23;
47
48 vector<int> ans[N];
49
50 int main() {
51

```

```

51
52     int test;
53     cin >> test;
54     while (test--) {
55         int n, k;
56         cin >> n >> k;
57
58         if (ans[n].empty()) {
59             // Dựng đồ thị de Bruijn n - 1 chiều
60             vector<vector<Edge>> g = build_graph(n - 1);
61
62             // Tìm chu trình Euler trên đồ thị vừa dựng
63             list<Edge> seq = euler_walk(0, g);
64
65             // Sinh dãy de Bruijn n chiều từ chu trình Euler tìm được
66             int cur = 0;
67             ans[n].clear();
68             for (auto it = seq.begin(); it != seq.end(); ++it) {
69                 cur = ((cur << 1) + it->weight) & max((1 << n) - 1, 1);
70                 ans[n].push_back(cur);
71             }
72         }
73
74         cout << ans[n][k] << "\n";
75     }
76     return 0;
}

```

## Vietnam TST 2017 - Problem 2 - Day 2

### Tóm tắt đề

Có  $N \leq 1000$  đoạn thẳng song song với trục tọa độ trên mặt phẳng. Mỗi đoạn được biểu diễn bởi hai điểm  $(x, y)$  ( $x, y \in \mathbb{Z}, -1000 \leq x, y \leq 1000$ ). In ra cách vẽ sao cho có thể tô màu tất cả các đoạn thẳng và số lần phải nhấc bút là ít nhất.

### Lời giải

Ta xem mỗi điểm trên mặt phẳng như một cạnh trên đồ thị vô hướng. Thêm 1 đỉnh ảo vào đồ thị. Với mỗi đỉnh bậc lẻ trong đồ thị ban đầu, nối đỉnh đó với đỉnh ảo vừa thêm. Khi này đồ thị ta đang có gồm nhiều thành phần liên thông, mỗi thành phần liên thông là một đồ thị Euler. Tìm chu trình Euler trên từng thành phần liên thông rồi xóa đỉnh ảo và các cạnh ảo đã thêm khỏi chu trình tìm được, ta thu được cách vẽ sao cho số lần nhấc bút là ít nhất.

### Cài đặt mẫu

```

#include <bits/stdc++.h>

using namespace std;

const int N = 1002;

```



```

struct Point {
    int x, y;

    Point(int _x, int _y): x(_x), y(_y) {}
};

struct Edge {
    Point source, target;
    int id;
    bool fake;

    Edge(int x, int y, int u, int v, int _id, bool _fake = false):
        source(x, y),
        target(u, v),
        id(_id),
        fake(_fake) {}
};

struct Graph {
    int edges;
    vector<vector<vector<Edge>>> graph;

    Graph() {
        edges = 0;
        graph.clear();
        graph.resize(N * 2);
        for (int i = 0; i < N * 2; ++i)
            graph[i].resize(N * 2);
    }

    vector<vector<Edge>>& operator[](int i) {
        return graph[i];
    }

    void addEdge(int x, int y, int u, int v, bool fake = false) {
        graph[x][y].emplace_back(x, y, u, v, edges, fake);
        graph[u][v].emplace_back(u, v, x, y, edges, fake);
        ++edges;
    }

    void addFakeEdges() {
        for (int i = 0; i < N * 2; ++i)
            for (int j = 0; j < N * 2; ++j) {
                if (graph[i][j].size() % 2 != 0) {
                    // Nối đỉnh bậc lẻ với đỉnh ảo
                    addEdge(i, j, 0, 0, true);
                }
            }
    }

    void addLine(int x1, int y1, int x2, int y2) {

```

```

    if (x1 > x2) swap(x1, x2);
    if (y1 > y2) swap(y1, y2);

    if (x1 == x2) { // nếu đoạn thẳng song song với trục Oy
        for (int i = y1; i < y2; ++i) {
            addEdge(N + x1, N + i, N + x2, N + i + 1);
        }
    } else { // nếu đoạn thẳng song song với trục Ox
        for (int i = x1; i < x2; ++i) {
            addEdge(N + i, N + y1, N + i + 1, N + y2);
        }
    }
}

};

int n; // Số đoạn thẳng trong đề
vector<bool> avail;
Graph g = Graph();
vector<vector<Point>> ans;

ostream& operator<<(ostream& out, Point p) {
    out << (p.x - N) << " " << (p.y - N);
    return out;
}

// Hàm euler_walk như trên
list<Edge> euler_walk(Point u, Graph &g) {
    list<Edge> ans;

    while (!g[u.x][u.y].empty()) {
        Edge e = g[u.x][u.y].back();
        g[u.x][u.y].pop_back();
        if (!avail[e.id]) continue;
        avail[e.id] = false;
        ans.emplace_back(e);
        u = e.target;
    }

    auto it = --ans.end();
    while (it != ans.begin())
    {
        list<Edge>::iterator tit = it;
        --tit;
        ans.splice(it, euler_walk(it->source, g));
        it = tit;
    }

    return ans;
}

int main() {

```

```

// Khởi tạo mảng đánh dấu cạnh đã đi qua hay chưa
avail.clear();

// Đọc đầu vào
cin >> n;
for (int i = 0; i < n; ++i) {
    int x, y, u, v;
    cin >> x >> y >> u >> v;
    g.addEdge(x, y, u, v);
}

// Thêm cạnh ảo vào đồ thị
g.addFakeEdges();

avail.assign(g.edges, true);
ans.clear();

// Duyệt qua tất cả các TPLT trong đồ thị
for (int i = 0; i < N * 2; ++i) {
    for (int j = 0; j < N * 2; ++j) {
        if (!g[i][j].empty()) {
            // Tìm chu trình Euler trên TPLT đang xét
            list<Edge> cycle = euler_walk(Point(i, j), g);

            // Chia chu trình tìm được thành các nét vẽ
            // riêng biệt dựa vào các cạnh ảo
            vector<Point> stroke;
            stroke.clear();
            for (list<Edge>::iterator it = cycle.begin(); it != cycle.end(); ++it) {
                if (it->fake) {
                    if (!stroke.empty()) {
                        ans.push_back(stroke);
                        stroke.clear();
                    }
                } else {
                    if (stroke.empty()) {
                        stroke.push_back(it->source);
                    }
                    stroke.push_back(it->target);
                }
            }
            if (!stroke.empty()) ans.push_back(stroke);
        }
    }
}

cout << ans.size() << "\n";
for (vector<Point> v : ans) {
    cout << v.size() << "\n";
    for (Point i : v) cout << i << "\n";
}

```

```
161 |  
162 |         return 0;  
    |     }
```

Được cung cấp bởi [Wiki.js](#)