**Contents**

Contents

[Graph 1](#_Toc171251940)

[Find-Union 1](#_Toc171251941)

[Park 1](#_Toc171251942)

[Connected Zones 6](#_Toc171251943)

[Connectivity 7](#_Toc171251944)

[Spanning Tree 8](#_Toc171251945)

[Minimal Spanning Tree 10](#_Toc171251946)

[Spanning Forest 12](#_Toc171251947)

[Minimal Spanning Forest 15](#_Toc171251948)

[Cycle 17](#_Toc171251949)

[Cycle In Connected Zone 18](#_Toc171251950)

[Shortest Cycle 22](#_Toc171251951)

[Net 25](#_Toc171251952)

[Water Valves 27](#_Toc171251953)

[Bridges 28](#_Toc171251954)

[Intersection Of Spanning Trees 31](#_Toc171251955)

[Joint Vertex 31](#_Toc171251956)

[Connections 33](#_Toc171251957)

[IT Teams 36](#_Toc171251958)

[Ants 37](#_Toc171251959)

[Subgraph 38](#_Toc171251960)

[Werewolf 42](#_Toc171251961)

# Graph

#### Định nghĩa

Một *đồ thị hữu hạn* G = (V, E) bao gồm tập V gồm n đỉnh mã số từ 1..n và tập cạnh E với các cặp đỉnh (x, y) nối hai đỉnh x và y. Nếu các cạnh (x, y) đều có chiều đi từ đỉnh x đến đỉnh y, x → y thì *đồ thị là có hướng;* ngược lại, khi không quy định hướng đi thì ta thu được *đồ thị vô hướng*. Mỗi cạnh (x, y) của đồ thị vô hướng cho phép ta đi từ đỉnh x đến đỉnh y, x → y, hoặc ngược lại, từ đỉnh y đến đỉnh x, x ← y giống như quy định trong giao thông về đường một hoặc hai chiều. Trong đồ thị có hướng ta dùng thuật ngữ *cung* (x, y) thay cho *cạnh.* Trong *đơn đồ thị,* người ta quy định giữa hai đỉnh bất kỳ x và y có không quá 1 cạnh (cung).

A diagram of a network

Description automatically generatedA diagram of a number

Description automatically generated

A diagram of a network

Description automatically generatedA diagram of a network

Description automatically generated

Một *đường p trong* đồ thị là dãy đỉnh , trong đó là đỉnh *xuất phát*, là *đỉnh kết thúc*, k là *chiều dài* của đường p. Nếu có hai đỉnh trong p trùng nhau thì đường đi gọi là *tự cắt,* ngược lại thì ta có đường *đơn*. Trong đồ thị vô hướng ta luôn coi hai đỉnh kề nhau trên đường p là khác nhau. Tùy theo nội dung của từng bài toán tin, người ta quy định là *có* hay *không có* đường đi từ đỉnh x đến chính nó, Nếu đỉnh cuối của đường trùng với điểm đầu của đường thì ta có một *chu trình*. Như vậy, chu trình ngắn nhất trong đồ thị vô hướng phải chứa tối thiểu 3 đỉnh và mọi tập gồm ba đỉnh trong đồ thị vô hướng luôn luôn tạo thành một chu trình.

Trong sách này, nếu không ghi chú gì thêm, ta ngầm hiểu các đồ thị được cho là *đơn đồ thị hữu hạn.*

Đồ thị là công cụ hữu hiệu được sử dụng để biểu diễn và xử lý nhiều bài toán tin.

## Find-Union

*Tìm-Gộp* là kỹ thuật dùng để quản lý hợp của các tập rời nhau. Ta minh họa kỹ thuật này qua bài toán sau:

## Park

*Cô giáo dẫn các em nhỏ đi chơi công viên. Lớp có n em mã số lần lượt từ 1 đến n. Vào công viên, để các bạn nhỏ khỏi lạc nhau, cô giáo quy định chia nhóm như sau:*

*Cô lần lượt đọc mã số của từng cặp hai bạn. Hai bạn đó sẽ thuộc cùng nhóm. Dĩ nhiên, nếu bạn A và B cùng nhóm, bạn B và C cùng nhóm thì cả ba bạn A, B và C là cùng nhóm. Cô cũng quy định rằng bạn nào có số hiệu nhỏ nhất trong nhóm sẽ làm nhóm trưởng. Sau m lần ghép cặp như vậy thì các em được chia thành mấy nhóm, hãy liệt kê mã số của các bạn trong từng nhóm.*

*Input: text file PARK.INP*

* *Dòng đầu tiên: hai số nguyên dương n và m,*
* *Dòng thứ i trong số m dòng tiếp theo: mỗi dòng hai số nguyên dương A B là mã số của hai bạn được ghép vào cùng nhóm.*

*Output: Hiển thị trên màn hình*

* *Dòng đầu tiên: k − số lượng nhóm*
* *Tiếp đến là k dòng, mỗi dòng liệt kê danh sách một nhóm.*

#### Ví dụ

|  |  |  |
| --- | --- | --- |
| PARK.INP | OUTPUT | *n = 8 em nhỏ, 5 lần ghép cặp*  *Em số 2 ghép cặp với em số 5*  *Em số 3 ghép cặp với em số 7. . .*  *3 nhóm.*  A group of circles with numbers  Description automatically generated |
| 8 5  2 5  3 7  8 6  5 1  4 3 | 3  1 2 5  3 4 7  6 8 |

#### Algorithm

Điều lý thú trong kỹ thuật này là ở điểm nhóm trưởng chính là em mang số hiệu nhỏ nhất trong nhóm. Lúc đầu, chưa ghép cặp thì mỗi em tạo thành một nhóm riêng. Như vậy, lúc đầu ta có n nhóm. Vì mỗi nhóm chỉ có duy nhất 1 em nên chính em đó là nhóm trưởng của chính mình. Ta mường tượng mỗi bạn x phải bám vai bạn đứng trước y trong cùng nhóm. Theo quy định của cô giáo thì y < x. Riêng nhóm trưởng, do có số hiệu nhỏ nhất nên tự bám vai mình.

Ta sử dụng mảng nguyên một chiều d[1..n] để quản lý các nhóm. Ta gán d[y] = x nếu bạn y phải bám vai bạn x là bạn có số hiệu nhỏ hơn mình: x ← y nếu y < x. Lúc đầu ta khởi trị d[i] = i; i = 1..n: i ← i, với ý nghĩa: lúc đầu, mỗi bạn tạo thành một nhóm 1 người và tự mình làm nhóm trưởng, do đó mỗi bạn tự bám vai chính mình. Mảng d được gọi là *mảng tham chiếu* hoặc *mảng trỏ ngược*. Như ta sẽ thấy sau này, nhờ d ta có thể nhanh chóng tìm được nhóm trưởng của bất kỳ em nào đồng thời liệt kê được các thành viên trong mỗi nhóm.

A group of circles with numbers and arrows

Description automatically generated

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Mỗi khi cô giáo yêu cầu ghép cặp (x, y) thì ta hiểu là: nhóm có em x cần hợp (gộp) với nhóm có em y. Để ghép cặp (x, y) ta thực hiện các bước sau đây:

*Bước 1.* Tìm nhóm trưởng x' của nhóm có em x

*Bước 2.* Tìm nhóm trưởng y' của nhóm có em y

*Bước 3.* Quyết định xem ai sẽ là nhóm trưởng của nhóm gộp mới này. Dễ hiểu là nhóm trưởng mới sẽ là em có số hiệu nhỏ nhất trong 2 nhóm trưởng x' và y', tức là min(x',y'). Sau đó ta gộp 2 nhóm theo kỹ thuật tham chiếu: Giả sử x' < y'. Ta chỉ việc gán d[y'] = x', y' → x'. Ý nghĩa của việc này là: cô giáo quy định nhóm trưởng y' phải bám vai nhóm trưởng x'. Ta suy ra ngay rằng mọi em trong nhóm y' đi theo nhóm trưởng y', mà y' lại đi theo x' nên mọi em trong nhóm y' sẽ đi theo x'. Nếu x' > y' ta gán d[x'] = y', x' → y' với ý nghĩa tương tự như trên. Nếu x' = y' thì có nghĩa là x và y hiện ở trong cùng một nhóm (vì có cùng nhóm trưởng) nên ta không phải làm gì.

Bạn quan sát lần lượt các hình dưới đây để phát hiện ra quy luật cập nhật từng cặp học sinh theo yêu cầu của cô giáo.

A group of circles with numbers

Description automatically generated

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | [2] | [3] | [4] | **[5]** | [6] | [7] | [8] |
|  | 1 | 2 | 3 | 4 | **2** | 6 | 7 | 8 |

A group of circles with numbers and arrows

Description automatically generated

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | [2] | [3] | [4] | [5] | [6] | **[7]** | [8] |
|  | 1 | 2 | 3 | 4 | 2 | 6 | **3** | 8 |

A diagram of circles and arrows

Description automatically generated

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | [2] | [3] | [4] | [5] | [6] | [7] | **[8]** |
|  | 1 | 2 | 3 | 4 | 2 | 6 | 3 | **6** |

A diagram of circles and arrows

Description automatically generated

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | **[2]** | [3] | [4] | [5] | [6] | [7] | [8] |
|  | 1 | **1** | 3 | 4 | 2 | 6 | 3 | 6 |

Để cập nhật cặp (5,1) ta để ý rằng nhóm trưởng của 5 là 2, vì d[5] = 2, nhóm trưởng của 1 là 1, vì d[1] = 1. Vì 1 < 2 nên ta cho nhóm trưởng 2 bám vào nhóm trưởng 1, tức là gán d[2] = 1, 2 → 1.

A diagram of circles and arrows

Description automatically generated

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | [2] | [3] | **[4]** | [5] | [6] | [7] | [8] |
|  | 1 | 1 | 3 | **3** | 2 | 6 | 3 | 6 |

Hàm Init khởi trị mảng d[1..n], d[i] = i với ý nghĩa lúc đầu mỗi em tạo thành một nhóm riêng biệt với nhóm trưởng là chính mình.

d = list(range(n+1))

Để ý rằng x sẽ là nhóm trưởng khi và chỉ khi d[x] = x. Từ đây ta cài đặt ngay được hàm Find(x): xác định nhóm trưởng của nhóm có em x như sau:

def Find(x): # nhóm trưởng của nhóm chứa x

while d[x] != x: x = d[x]

return x

Để gộp 2 nhóm: nhóm có em x và nhóm có em y ta lưu ý hai điểm sau đây:

* Rất có thể 2 em x và y hiện đã trong cùng một nhóm. Trường hợp này ta không phải làm gì. Hàm sẽ cho ra giá trị 0. Ta thấy x và y cùng nhóm khi và chỉ khi x và y có cùng nhóm trưởng, tức là Find(x) = Find(y).
* Nếu 2 em x và y thuộc 2 nhóm khác nhau, Find(x) ≠ Find(y), thì ta thực sự gộp 2 nhóm này và hàm Union cho ra giá trị 1.

def Union(x,y): # Gộp nhóm chứa x với nhóm chứa y

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

Việc cần làm cuối cùng là liệt kê thành viên của mỗi nhóm. Ta duyệt lại mảng d, mỗi khi tìm được một nhóm trưởng i theo hệ thức d[i] = i, do i là nhóm trưởng nên mọi thành viên của nhóm này đều có số hiệu bằng hoặc lớn hơn i. Do đặc điểm này nên ta chỉ cần duyệt tiếp mảng d kể từ j = i đến n, nếu nhóm trưởng của j là i thì ta hiển thị thành viên j.

Nhớ rằng lúc đầu ta có tổng cộng n nhóm. Mỗi khi ta gộp thực sự 2 nhóm, tức là khi Union(x,y) = 1 thì số lượng nhóm sẽ giảm đi 1. Nhờ nhận xét này ta dễ dàng tính được số nhóm tại thời điểm kết thúc.

Kết quả của thuật toán sử dụng kỹ thuật Find-Union không phụ thuộc vào trật tự duyệt các cặp ghép.

#### Độ phức tạp tính toán

Hàm Find thực hiện tối đa n bước tham chiếu x = d[x]. Hàm Union gọi hàm Find do đó có độ phức tạp O(n). Tổng cộng lại, với đồ thị n đỉnh, m cạnh ta có độ phức tạp O(nm): tuyến tính theo chiều dài input.

#### Ghi nhớ

Để vận dụng kỹ thuật Find-Union ta cần:

1. Init: Khai báo một mảng nguyên d, khởi trị d[i] = i với i = 1..n

2. Cài đặt hàm Find

3. Cài đặt hàm Union.

4. Sau khi thực hiện xong lượt ghép cặp cuối cùng, bạn nên chuyển mảng tham chiếu d sang dạng tham chiếu trực tiếp. Cụ thể là chuyển đỏi

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | [2] | [3] | [4] | **[5]** | [6] | [7] | [8] |
|  | 1 | 1 | 3 | 3 | **2** | 6 | 3 | 6 |

thành dạng sau: Thay d[5] = 2 (trỏ gián tiếp) thành d[5] = 1 (trỏ trực tiếp)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | [1] | [2] | [3] | [4] | **[5]** | [6] | [7] | [8] |
|  | 1 | 1 | 3 | 3 | **1** | 6 | 3 | 6 |

# Đặt lại trỏ trực tiếp

for i in range(1, n+1): d[i] = Find(i)

#### Chương trình

# Park: Kĩ thuật Find-Union

FN = 'PARK.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

def Park():

global d

with open(FN) as f:

data = list(map(int,f.read().split()))

print('Input:',data)

n, m = data[0], data[1]

print(n,'đỉnh,', m, 'cạnh')

d = list(range(n+1))

print('Init d:', d)

sn, j = n, 0 # sn: số nhóm

for i in range(m):

j += 2

sn -= Union(data[j], data[j+1])

# Đặt lại trỏ trực tiếp

r = range(1, n+1)

for i in r: d[i] = Find(i)

print(sn, 'nhóm')

k = 0

for i in r:

if d[i] == i: # i là nhóm trưởng

k += 1

print(' \* Nhóm', k, end = ': ')

for j in range(i,n+1):

if d[j] == i: print(j, end = ' ')

print()

# APPLICATION

Park()

print(" T h e E n d")

#### Output

Input: [8, 5, 2, 5, 3, 7, 8, 6, 5, 1, 4, 3]

8 đỉnh, 5 cạnh

Init d: [0, 1, 2, 3, 4, 5, 6, 7, 8]

3 nhóm

\* Nhóm 1: 1 2 5

\* Nhóm 2: 3 4 7

\* Nhóm 3: 6 8

T h e E n d

## Connected Zones

*Cho đồ thị vô hướng G =* (*V, E*) *với n đỉnh V =* {1..n} *và m cạnh* (*x,y*) *nối đỉnh x với đỉnh y. Hãy tính số thành phần liên thông (TPLT) và liệt kê tập đỉnh trong mỗi TPLT của G.*

#### Understanding

Đây chính là nội dung của bài Park được trình bày qua ngôn ngữ đồ thị. Đồ thị vô hướng G được gọi là *liên thông* nếu từ một đỉnh bất kỳ i ta có thể đi theo một số cạnh liền kề nhau của đồ thị để đến một đỉnh j bất kỳ. Dãy các cạnh liền kề từ đỉnh i đến đỉnh j được gọi là một *đường* từ i đến j. Với đồ thị vô hướng, khi có đường từ đỉnh i đến đỉnh j thì ta cũng có đường từ đỉnh j đến đỉnh i. Nếu G không liên thông thì G được chia thành các *mảnh liên thông,* còn gọi là *thành phần liên thông (TPLT)*, gồm một số đỉnh và cạnh của G*.* Hãy tưởng tượng mỗi đỉnh của đồ thị như là một hạt cườm, cạnh nối hai đỉnh i và j chính là sợi dây nối hai hạt i và j. Khi đó, G là *liên thông* khi và chỉ khi ta cầm một hạt bất kỳ nhấc lên thì tất cả các hạt đều được nhấc theo. Nếu G không liên thông thì mỗi lần nhấc một hạt ta được một TPLT gồm một số hạt được liên kết với nhau. Ta đặt riêng mảnh đó ra, rồi nhấc một hạt bất kỳ trong số các hạt còn lại ta sẽ được TPLT thứ hai,…

#### Nhận xét

G là liên thông khi và chỉ khi số TPLT của G là 1.

Đồ thị trong ví dụ dưới đây gồm 3 TPLT (1, 5, 2), (4, 3, 7) và (8, 6).

|  |  |  |
| --- | --- | --- |
| GRAPH.INP | OUTPUT | *3 TPLT*: {1, 2, 5}, {3, 4, 7} và {6, 8}  A group of circles with numbers  Description automatically generated |
| 8 5  2 5  3 7  8 6  5 1  4 3 | 3  1 2 5  3 4 7  6 8 |

#### Algorithm

#### Nhận xét

Số TPLT của G = số nhóm trưởng.

Để tính số TPLT của G ta thực hiện hai bước sau:

Bước 1. Khởi trị d[i] = i; i = 1..n; gán k = n dùng để đếm số TPLT.

Bước 2. Đọc dần từng cạnh (x,y) và thực hiện toán tử k -= Union(x,y);

k sẽ là số TPLT của G.

Lưu ý rằng hàm Union(x,y) cho giá trị 1 nếu TPLT chứa đỉnh x được gộp thực sự với TPLT chứa đỉnh y, ngược lại, Union(x,y) = 0 nếu hai đỉnh x và y hiện đang cùng có mặt trong một TPLT (do đó không cần thiết gộp hai thành phần này). Như vậy ý nghĩa của giá trị ra của hàm Union(x,y) là giảm số TPLT của G. Dĩ nhiên, sau khi thực hiện hàm Union(x,y) thì hai đỉnh x và y sẽ thuộc cùng một TPLT.

#### Độ phức tạp

O(nm) với n là số đỉnh, m là số cạnh của đồ thị

Ghi nhớ

Sau khi vận dụng kỹ thuật Find-Union mảng tham chiếu d cho ta các thông tin sau đây:

Với mỗi đỉnh i = 1..n

* d[i] = i khi và chỉ khi đỉnh i là "nhóm trưởng", tức là đỉnh đại diện cho một thành phần liên thông trong đồ thị;
* Số lượng thành phần liên thông trong G = số lượng nhóm trưởng = số lượng các đỉnh i thỏa điều kiện d[i] = i;
* Hai đỉnh i và j thuộc cùng một thành phần liên thông khi và chỉ khi Find(i) = Find(j);
* Mỗi đỉnh i thuộc một thành phần liên thông duy nhất.

## Connectivity

*Cho đồ thị vô hướng G = (V, E) với n đỉnh V = {1..n} và m cạnh (x,y) nối đỉnh x với đỉnh y. Hãy hiển thị 1 nếu đồ thị G liên thông, 0 nếu G không liên thông.*

Ví dụ

|  |  |  |
| --- | --- | --- |
| graph.inp | Output | *8 đỉnh và 5 cạnh.*  *G không liên thông vì có hơn 1 TPLT (3)*  A group of circles with numbers  Description automatically generated |
| 8 5  2 5  3 7  8 6  5 1  4 3 | 0 |

#### Algorithm

Đồ thị G liên thông khi và chỉ khi số TPLT bằng 1.

Độ phức tạp: O(nm), n là số đỉnh, m là số cạnh của G.

## Spanning Tree

*Cây khung* (còn gọi là *cây bao trùm*) của một đồ thị liên thông vô hướng n đỉnh, m cạnh là cây gồm n đỉnh với số cạnh tối thiểu bảo toàn tính liên thông của đồ thị.

Nhận xét: *Cây khung của đồ thị n đỉnh, m cạnh có đúng n đỉnh và n−1 cạnh.*

Về mặt trực quan, nếu ta mường tượng đồ thị như một bộ đèn chùm treo trên trần nhà với mỗi bóng đèn là một đỉnh, mỗi thanh cứng nối hai bóng đèn là một cạnh của đồ thị thì yêu cầu đặt ra là có thể bỏ bớt một số cạnh mà các bóng đền không rơi xuống đất được không.

Số cạnh tối thiểu còn lại giữ cho các bóng đèn không rơi được gọi là *cây khung.*

Có hai thuật toán, hai cách tiếp cận để tìm cây khung là thuật toán Prim và thuật toán Kruskal.

* Thuật toán Prim có ý tưởng như vừa trình bày: Treo chùm đền rồi thử tháo dần từng thanh nối, Nếu tháo thanh nối (x,y) làm cho bóng đèn rơi ra thì lắp lại thanh nối đó. Độ phức tạp O(m), m là số cạnh của đồ thị
* Thuật toán Kruskal thao tác ngược lại với thuật toán Prim: Xuất phát từ đèn chùm rỗng, chỉ có một móc treo trên trần ta lắp dần các thanh nối hai bóng đèn (x,y). Nếu hai bóng này trước đó đã được lắp bới các thanh nối rồi, tức là x và y đã liên thông với nhau thì bỏ qua thanh nối (x,y).

#### Algorithm

Pha 1. Đọc và chuẩn hóa dữ liệu. Ta dùng một từ điển e = {key:val} của đồ thị. key là cạnh (x,y), và key nhận giá trị 1 nếu (x,y) là cạnh của đồ thị, nhận giá trị 2 nếu (x,y) là cạnh thuộc cây khung. Ta cũng quy ước là chỉ nhận cạnh dạng (x, y), x < y để dễ dàng kiểm tra các cạnh trùng lặp. Độ phức tạp O(m).

Pha 2. Gọi thuật toán Kruskal để xác định cây khung

#### Thuật toán Kruskal

Duyệt các cạnh: nếu cạnh (x*,y*) không tạo thành chu trình với cây khung thì cập nhật e[(x,y)] = 2.

Điều kiện để cạnh (x*,y*) không tạo thành chu trình với cây khung là Union(x*,y*) = 0.

Độ phức tạp O(nm).

|  |  |  |
| --- | --- | --- |
| graph.inp | Output | *Input: Đồ thị có 8 đỉnh và 11 cạnh.*  *Output: 7 cạnh của cây khung (tô đậm)*  *A diagram of a network  Description automatically generated*Trật tự: (1,2), (1,3), (1,6),  (2,4), (4,5), (5,7), (5,8) |
| 8 11  1 2  1 3  1 6  2 4  2 6  3 4  4 5  5 6  5 7  5 8  7 8 | 1 2  1 3  1 6  2 4  4 5  5 7  5 8 |

#### Program

Hàm SP() cho ra hai giá trị b và e. b = True nếu đồ thị có cây khung, b = False nếu đồ thị không có cây khung; e là từ điển lưu trữ các cạnh.

Đồ thị hữu hạn có cây khung khi và chỉ khi đồ thị là liên thông. Như vậy, khi số TPLT > 1 thì đồ thị không liên thông và do đó không có cây khung.

# Spanning Tree (Algorithm Kruskal)

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

# Đọc dữ liệu input, tổ chức đồ thị G = (V,E) số đỉnh n, tập cạnh e

def ReadInput():

with open(FN) as f:

data = list(map(int,f.read().split()))

n, m = data[0], data[1] # n đỉnh, m cạnh

"""

Từ điển cạnh e = {key:val}

key: cạnh (x,y)

val: 1 thuộc đồ thị, 2: thuộc cây khung

"""

e, j = {}, 0

for i in range(m):

j += 2

if data[j] == data[j+1]: continue

if data[j] < data[j+1]: c = (data[j], data[j+1])

else: c = (data[j+1], data[j])

if not c in e: e[c] = 1

return n, e

def SP(): # spanning tree

global d

n, e = ReadInput() # n: số đỉnh, e: tập các cạnh

if len(e) < n-1: return False, e

#print(n)

#print(e)

d = list(range(n+1))

count = 0 # đếm số cạnh trong cây khung

for c in e:

#print(c)

if Union(c[0], c[1]):

e[c] = 2

count += 1

if count == n-1: break

return count == n-1, e

# APPLICATION

b, e = SP()

print(' Spanning tree')

if not b: print(' No')

else:

for c in e:

if e[c]== 2: print(c)

print(" T h e E n d")

#### Output

Spanning tree

(1, 2)

(1, 3)

(1, 6)

(2, 4)

(4, 5)

(5, 7)

(5, 8)

T h e E n d

## Minimal Spanning Tree

*Cho đồ thị G vô hướng và liên thông với n đỉnh và m cạnh, cạnh (u,v) có trọng số p(u,v) là một số dương. Cây khung cực tiểu (còn gọi là cây bao trùm ngắn nhất) của G là cây khung với tổng trọng số của các cạnh trong khung là nhỏ nhất.*

#### Algorithm

Thuật toán tìm cây khung cực tiểu hoạt động giống thuật toán tìm cây khung với một điểm khác biệt duy nhất là *duyệt các cạnh theo trật tự tăng dần của trọng số*.

*Bước 1.* Sắp các cạnh tăng theo trọng số;

*Bước 2.* Duyệt các cạnh theo trật tự đã sắp: nếu cạnh (u,v) không tạo thành chu trình với cây khung thì nạp (u,v) vào cây khung. Điều kiện để cạnh (u,v) không tạo thành chu trình với cây khung là Union(u,v) = 0.

#### Độ phức tạp

Max của hai giá tri:

O(nm) chi phí cho việc duyệt

O(m log(m)) chi phí cho sắp tăng m cạnh

Tùy theo các ứng dụng, ta có thể sử dụng trọng số để biểu thị độ dài đường đi giữa hai đỉnh hoặc chi phí phải trả khi đi từ đỉnh này đến đỉnh kia.

Ta mô tả cạnh dưới dạng từ điển e, mỗi phần tử có dạng (x,y):p với (x,y) là cạnh, x < y và p là trọng số của cạnh (x,y): e[(x,y)] = p. Ngoài ra, ta sử dụng danh sách mst để chứa các cạnh được chọn vào cây khung cực tiểu.

|  |  |  |
| --- | --- | --- |
| graph.inp | Output | *Đồ thị có 8 đỉnh và 11 cạnh dạng x y p (trọng số)*  *7 cạnh của cây khung cực tiểu (tô đậm)*  *Tổng trọng số min: 30*  A diagram of a network  Description automatically generated |
| 8 11  1 2 2  1 3 7  1 6 9  2 4 4  2 6 8  3 4 6  4 5 10  5 6 5  5 7 4  5 8 2  7 8 3 | 5 8  1 2  7 8  2 4  5 6  3 4  2 6  30 |

#### Program

# Mnimal Spanning Tree (Algorithm Kruskal)

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

# Đọc dữ liệu input, tổ chức đồ thị G = (V,E) số đỉnh n, tập cạnh e

def ReadInput():

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

print(n, 'đỉnh', m, 'cạnh')

"""

dict of edges e{(x,y):d}

"""

e = {}

for i in range(m):

x, y, p = map(int,f.readline().strip().split())

if x == y: continue

c = (x, y) if x < y else (y, x)

if not c in e: e[c] = p

return n, e

def MinSP(): # spanning tree

global d

n, e = ReadInput() # n: số đỉnh, e: tập các cạnh

# Sắp tăng e.keys theo trọng số

k = sorted(list(e.keys()), key=lambda x: e[x])

mst = [] # list of edges in minimal spanning tree

d = list(range(n+1)) # mảng tham chiếu

for edg in k:

if Union(edg[0],edg[1]) > 0: mst.append(edg)

return mst, e

# APPLICATION

mst, e = MinSP()

print(' Minimal Sanning tree:')

sumd = 0 # Total weight

for c in mst:

print(c)

sumd += e[c]

print('Total weight:', sumd)

print(" T h e E n d")

#### Output

Minimal Sanning tree:

(1, 2)

(5, 8)

(7, 8)

(2, 4)

(5, 6)

(3, 4)

(2, 6)

Total weight: 30

T h e E n d

## Spanning Forest

*Cho G là một đồ thị vô hướng gồm n đỉnh và m cạnh. Hãy xác định các cây khung trong mỗi mảnh liên thông của G. Tập hợp các cây khung đó được gọi là rừng khung của đồ thị G. Hãy liệt kê các cây khung thành phần của rừng khung.*

|  |  |
| --- | --- |
| graph.inp | *Đồ thị có 8 đỉnh và 8 cạnh.*  *Rừng khung gồm hai cây khung (tô đậm)*  *Cây khung liên thông thuộc đỉnh 1 gồm 2 cạnh:*  (1,2), (1,5).  *Cây khung liên thông thuộc đỉnh 3 gồm 4 cạnh:*  A diagram of a network  Description automatically generated(3,4), (3,7), (3,8), (4,6). |
| 8 8  1 2  3 4  3 7  1 5  3 8  4 6  6 8  2 5 |

#### Algorithm

Để ý rằng dù đồ thị (vô hướng) liên thông hay không liên thông thì thủ tục SP như mô tả trong các bài trước đều cho ta một danh sách các cạnh được chọn vào cây khung cần tìm. Danh sách này được ghi trong từ điển e {cạnh:mark}. Nếu đồ thị có trên một TPLT thì ta tách các cạnh của e thành các cây con trong rừng cần tìm, mỗi cạnh sẽ được chỉ định rõ là thuộc cây khung của TPLT nào. Như trong các bài trước, ta quy ước mã số mỗi TPLT là số hiệu nhỏ nhất của đỉnh thuộc TPLT đó (tức là "nhóm trưởng" của nhóm đó). Cạnh (x,y), x < y thuộc TPLT i khi và chỉ khi Find(x) = i. Vì x và y là cạnh trong một cây khung thành phần nên ta luôn có Find(x) = Find(y). Đó là lý do ta chỉ cần xác định nhóm trưởng cho một trong hai đỉnh x hoặc y.

Lưu ý rằng, để tiện xử lý, ta thêm cho dữ liệu mô tả cạnh một trường kiểu nguyên mark để ghi nhận cạnh này thuộc mảnh liên thông nào. Khi khởi tạo ta gán trị mark = n+1 cho mọi cạnh của đồ thị.

Thuật toán Rừng khung SF gồm các bước sau đây:

* Bước 1. Đọc dữ liệu vào từ điển e. Độ phức tạp O(m)
* Bước 2. Dùng thuật toán Kruskal đánh dấu các cạnh được chọn cho rừng khung. Độ phức tạp O(nm)
* Bước 3. Phân loại cạnh trong rừng khung theo số hiệu của TPLT. Độ phức tạp O(m)
* Bước 4. Sắp tăng trường khóa keys của e theo số hiệu cạnh thu được tại bước 3. Độ phức tạp O(mlog(m))
* Bước 5. Thông báo kết quả: Mỗi TPLT có một cây khung. Độ phức tạp O(m)

#### Độ phức tạp

Nếu đồ thị có n đỉnh và m cạnh thì độ phức tạp tính toán là giá trị max của hai thông số:

* Xác định các cạnh khung bằng thuật toán Kruskal cần độ phức tạp O(nm)
* Sắp tăng m cạnh cần thời gian O(mlog(m)).

#### Nhận xét

*Tổng số cạnh trong các cây khung của rừng khung là* số đỉnh − số TPLT.

Thật vậy, giả sử đồ thị có n đỉnh và k TPLT. Gọi N(i) là số đỉnh trong mảnh liên thông thứ i; i = 1..k, T là tổng số cạnh trong các cây của rừng khung. Để ý rằng n đỉnh được phân hoạch vào k mảnh liên thông (không giao nhau) và cây khung của mảnh liên thông thứ i có N(i)i−1; i = 1..k. Ta có

N(1) + N(2) + ... + N(k) = n

T = (N(1)-1) + (N(2)−1) +…+ (N(k)-1) = n - k.

#### Program

# Spanning Forest

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

# Đọc dữ liệu input, tổ chức đồ thị G = (V,E) số đỉnh n, tập cạnh e

def ReadInput():

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

print(n,'đỉnh', m, 'cạnh')

n1 = n + 1 # đánh dấu cạnh không thuộc cây khung

"""

dict e{(x,y):t}, cạnh (x,y), TPLT t

"""

e = {}

for i in range(m):

x, y = map(int,f.readline().strip().split())

if x == y: continue

c = (x, y) if x < y else (y, x)

e[c] = n1

return n, e

def SF(): # spanning forest

global d

n, e = ReadInput() # n: số đỉnh, e: tập các cạnh

d = list(range(n+1)) # Khởi tạo mảng tham chiếu (trỏ ngược)

count = 0 # đếm số cạnh trong cây khung

for c in e:

if Union(c[0],c[1]):

count += 1

e[c] = 1 # đánh dấu cạnh thuộc cây khung

if count == n-1: break

for i in range(1, n+1): d[i] = Find(i) # đặt lại trỏ trực tiếp

for c in e:

if e[c] == 1: e[c] = d[c[0]]

k = sorted(list(e.keys()), key=lambda x: e[x])

m = len(k)

v = 0

for i in range(m):

if e[k[i]] > n: break

if e[k[i]] != v:

v = e[k[i]]

print('Spanning Tree of vertex',v)

for j in range(i,m):

if e[k[j]] == v: print(k[j])

else: break

# APPLICATION

SF()

print(" T h e E n d")

#### Output

Spanning Tree of vertex 1

(1, 2)

(1, 5)

Spanning Tree of vertex 3

(3, 4)

(3, 7)

(3, 8)

(4, 6)

T h e E n d

## Minimal Spanning Forest

Rừng khung cực tiểu được xác định như rừng khung với phép duyệt các cạnh theo trật tự tăng dần của trọng số cạnh.

Mỗi cạnh trong bài này được mô tả như trong bài Cây khung cực tiểu: e = {(x,y):p}, trong đó (x,y) là cạnh, p là trọng số của cạnh .

|  |  |
| --- | --- |
| graph.inp | *Đồ thị có 8 đỉnh và 8 cạnh dạng x y p.*  *Rừng gồm hai cây khung (tô đậm)*  *A diagram of a network  Description automatically generated*  *Cây khung liên thuộc đỉnh 1 gồm 2 cạnh: (1,2), (2,5)*  *và trọng số = 3+5 = 8.*  *Cây khung liên thuộc đỉnh 3 gồm 4 cạnh: (3,7), (3,8), (4,6), (6,8)*  *và trọng số = 7+2+5+3 = 17.*  *Tổng trọng số min = 8+17 = 25.* |
| 8 8  1 2 3  1 5 9  2 5 5  3 4 9  3 7 7  3 8 2  4 6 5  6 8 3 |

#### Algorithm

*Bước 1.* Đọc từ input file các giá trị: số đỉnh n, số cạnh m và tiếp đến là m cạnh, mỗi cạnh 3 giá trị đỉnh đầu x, đỉnh cuối y và trọng số p vào từ điển e {(x,y):p}. Độ phức tạp O(m)

*Bước 2.* Sắp tăng e.keys theo trọng số p. Độ phức tạp O(mlog(m))

*Bước 3.* Tạo một bản sao ec theo e.keys: sửa trường p thành trường mark để đánh dáu các cạnh khung. Độ phức tạp O(m)

Bước 4. Dùng thuật toán Kruskal xác định các cạnh thuộc rừng khung. Độ phức tạp O(nm)

Bước 5. Phân loại cạnh trong rừng khung theo số hiệu của TPLT. Độ phức tạp O(m)

*Bước 6.* Sắp tăng ec.keys theo số hiệu thu được tại Bước 5. Độ phức tạp O(mlog(m))

*Bước 7.* Thông báo kết quả theo từng TPLT. Độ phức tạp O(m)

#### Program

# Minimal Spanning Forest

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

# Đọc dữ liệu input, tổ chức đồ thị G = (V,E) số đỉnh n, tập cạnh e

def ReadInput():

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

print(n,'đỉnh', m, 'cạnh')

"""

dict e{(x,y):p}, cạnh (x,y), trọng số p

"""

e = {}

for i in range(m):

x, y, p = map(int,f.readline().strip().split())

if x == y: continue

c = (x, y) if x < y else (y, x)

e[c] = p # trọng số

return n, e

def MSF(): # Minimal Spanning Forest

global d

n, e = ReadInput() # n: số đỉnh, e: tập các cạnh (x,y):p

kp = sorted(list(e.keys()), key=lambda x: e[x])

ec = {} # lấy bản sao của e

# sửa trường p thành mark = n2

n1 = n + 1

for c in kp: ec[c] = n1

d = list(range(n+1)) # Khởi tạo mảng tham chiếu (trỏ ngược)

count = 0 # đếm số cạnh trong cây khung

# Kruskal

for c in ec:

if Union(c[0],c[1]):

count += 1

ec[c] = 1 # đánh dấu cạnh thuộc cây khung

if count == n-1: break

for i in range(1, n+1): d[i] = Find(i) # đặt lại trỏ trực tiếp

for c in ec:

if ec[c] == 1: ec[c] = d[c[0]]

# Sorted by TPLT

k = sorted(list(ec.keys()), key=lambda x: ec[x])

# Thông báo kết quả

m = len(k)

print('Minimal Spanning Forest:')

v = tt = 0 # tt: tổng trọng số của các cây khung thành phần

for i in range(m):

if ec[k[i]] > n: break

if ec[k[i]] != v: # new spanning tree

t = 0 # partial weight

v = ec[k[i]]

print('Spanning Tree of vertex',v)

for j in range(i,m):

if ec[k[j]] == v: print(k[j]); t += e[k[j]]

else: print('Partial Weight:', t); tt += t; break

print('Total Weight:', tt)

# APPLICATION

MSF()

print(" T h e E n d")

#### Output

Minimal Spanning Forest:

Spanning Tree of vertex 1

(1, 2)

(2, 5)

Partial Weight: 8

Spanning Tree of vertex 3

(3, 8)

(6, 8)

(4, 6)

(3, 7)

Partial Weight: 17

Total Weight: 25

T h e E n d

## Cycle

*Cho đồ thị vô hướng G = (V, E) với n đỉnh V = {1..n} và m cạnh (x, y) nối đỉnh x với đỉnh y. Hãy hiển thị 1 nếu đồ thị G có chứa ít nhất một chu trình, 0 nếu G không có chu trình.*

*Ví dụ 1*

|  |  |  |
| --- | --- | --- |
| graph.inp | Output | *Đồ thị có 8 đỉnh và 5 cạnh.*  *Đồ thị không có chu trình: 0.*  A group of circles with numbers  Description automatically generated |
| 8 5  2 5  3 7  8 6  5 1  4 3 | 0 |

Ví dụ 2

|  |  |  |
| --- | --- | --- |
| graph.inp | Output | A diagram of a tree  Description automatically generatedĐồ thị coa 2 chu trình |
| 10 10  9 1  2 5  4 7  8 6  5 1  4 3  9 10  10 2  4 8  3 6 | 1 |

#### Algorithm

Nhận xét 1: *Đồ thị vô hướng G có chu trình khi và chỉ khi trong quá trình ghép cạnh ta gặp một cạnh (x, y) thoả tính chất x ≠ y và Union(x, y) = 0.*

Thật vậy, Union(x, y) = 0 chứng tỏ tại thời điểm trước khi thực hiện hàm Union(x, y), hai đỉnh x và y thuộc cùng một TPLT, nghĩa là có đường đi từ x đến y. Nay ta thêm cạnh (x, y) tức là ta có thể đi tiếp thêm một bước nữa từ x đến y, hoặc từ y đến x vì đồ thị là vô hướng. Như vậy là đã xuất hiện một chu trình từ x trở về x. Ta gọi TPLT này là *TPLT khép kín*.

Nhận xét 2. Ta biết nếu đồ thị hữu hạn, vô hướng thì số cạnh trong cây khung là c = n-k, trong đó n là số đỉnh của đồ thị, k là số thành phần liên thông. Gọi số cạnh khác nhau của đồ thị là m, thì khi m ≥ n thì chắc chắn đồ thị có chu trình. Thật vậy, vì c = n - k < n ≤ m, suy ra c < m. Do đó trong quá trình ghép cạnh ta sẽ gặp tình huống Union(x, y) = 0, tức là đồ thị có chu trình.

Độ phức tạp: O(*nm*) với *n* là số đỉnh, *m* là số cạnh của *G*.

#### Program

# Cyclic

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

# Đọc dữ liệu input, tổ chức đồ thị G = (V,E) số đỉnh n, tập cạnh e

def Cyclic():

global d

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

# print(n,m)

d = list(range(n+1)) # mảng tham chiếu

for i in range(m): # cạnh (x,y)

x, y = map(int,f.readline().strip().split())

# print(x,y)

if x == y: continue

if Union(x,y) == 0: return 1

return 0

# APPLICATION

print(Cyclic())

print(" T h e E n d")

## Cycle In Connected Zone

1. Chương trình trên chỉ đơn giản hiển thị 1: đồ thị có chu trình, hoặc 0: không có chu trình.

2. Trong trường hợp đồ thị có chu trình và bạn muốn chỉ ra một trong số các chu trình thì bạn có thể theo thuật toán sau đây

* 2.1 Đọc dữ liệu vào ma trận kề a, mỗi cạnh (x,y) sẽ tạo ra hai phép gán trị: a[x][y] = a[y][x] = 1
* Dùng kỹ thuật Find-Union phát hiện chu trình và TPLT khép kín cz chứa chu trình cuối cùng mà bạn phát hiện được. Ví dụ trên cho ta cz = [3, 4, 6, 7, 8].
* 2.2 Phỏng theo thuật toán quay lui trong bài Sợi chỉ Arian (trang 246, Chuyên Tin 10) tìm một đường đi v từ đỉnh x đến các đỉnh còn lại thuộc TPLT cz. Nếu đường đi v có len ≥ 3, cụ thể là v có dạng v = [..x..y,z] thì kiêm tra xem có tồn tại cạnh (x,z) hay không. Nếu có, thì ta sẽ nhận được chu trình

A drawing of a circle and a line

Description automatically generated

Gọi cz là tập các đỉnh trong TPLT khép kín, ta cần tìm chu trình trong cz, tức là tìm một đường khép kín vv = x-y…x, len(vv) > 3 chứa các đỉnh trong cz.

* Nếu len(cz) < 3 thì cz không chứa chu trình
* Nếu len(cz) = 3: thì bản thân cz sẽ tạo thành chu trình sau khi thêm phần tử đầu tiên vào cuối cz.
* Nếu len(cz) > 3 thì chắc chắn cz chứa một chu trình nào đó, ta tìm chu trình này theo tiếp cận backtracking như sau:

Xuất phát từ x = cz[0]

Tìm đường đi v bắt đầu từ x tới đỉnh z ≠ x trong cz. Giả sử v = [x, y,…,z]

Mỗi khi tìm được thêm đỉnh z ta kiểm tra xem từ z trở về trước có đỉnh x nào nối với z hay không. Nếu có đỉnh x thỏa: x xuất hiện trước y, y là đỉnh sát trước z, có cạnh (x,z) và len(v) ≥ 3 thì tức là xuất hiện chu trình (xem hình vẽ và hàm Goal).

Ví dụ

|  |  |  |
| --- | --- | --- |
| graph.inp | Output | A diagram of a tree  Description automatically generatedĐồ thị có 2 chu trình  Chương trình hiển thị chu trình  3 - 4 - 8 - 6 - 3 |
| 10 10  9 1  2 5  4 7  8 6  5 1  4 3  9 10  10 2  4 8  3 6 | 1  3 4 8 6 3 |

#### Algorithm Cycle

# shorted cycle in connected zone cz

def Cycle(cz): # backtracking

global visit, v

if len(cz) < 3: return []

if len(cz) == 3: cz + [cz[0]]

visit = bytearray([0]\*len(d))

v = [0]\*len(cz) # path

x = cz[0]

k = 0 # bước đi

v[k] = x # xuat phat tu dinh s

visit[x] = 1 # da tham dinh s

while True:

vv = Goal(cz)

if len(vv) > 3: return vv # den dich

if k < 0: return []

z = Next(cz, v[k]) # v[k] -> z ?

if z > 0: # tien

k += 1

v[k], visit[z] = z, 1

else: k -= 1 # lui

# Tìm đường đi từ x -> z

def Next(cz, x):

for z in cz:

if x != z and not visit[z] and a[x][z]:

return z

return -1

"""

Đã đến đích ?

xét các đỉnh đã thăm trong path v

Gọi z là đỉng cuối trong số các đỉnh trên

và y là đỉnh sát trước z

Nếu có cạnh (x,z), x trước y tức là có

chu trình

"""

def Goal(cz):

# xét các đỉnh x > 0 trong path v

vv = []

for x in v:

if x > 0: vv.append(x)

if len(vv) < 3: return []

# len(vv) >= 3: vv = [...z]

z = vv[-1]

for x in vv[:-2]: # cạnh (x,z) tạo chu trình

if a[x][z]:

i = vv.index(x)

return vv[i:]+[x]

return []

#### Độ phức tạp

Thuật toán Cycle nói trên tìm chu trình ngắn nhất trong TPLT cz với độ phức tạp

#### Program

# Cyclic in connected zone

# Nếu đồ thị có chu trình thì tìm 1 chu trình

# trong TPLT khép kín

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

""" Đọc dữ liệu vào ma trận kề a

Dùng kỹ thuật Find-Union để xác định TPLT chứa cycle

n: số đỉnh

cz: TPLT chứa cycle

"""

def ReadInput():

global n, a, d

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

# vertices vt

a = [[0]\*(n+1) for i in range(n+1)]

d = list(range(n+1)) # mảng tham chiếu

cz = []

xx = yy = 0

for i in range(m): # cạnh (x,y)

x, y = map(int,f.readline().strip().split())

if x == y: continue

a[x][y] = a[y][x] = 1

if Union(x,y) == 0: # gặp cycle

xx, yy = x, y # Ghi nhận cạnh khép kín cuối cùng

if xx == 0: return

# xác định TPL chứa cycle cz: connected zone

# đặt lại trỏ trực tiếp

for i in range(n+1): d[i] = Find(i)

i = d[xx]

for j in range(i, n+1):

if d[j] == i: cz.append(j)

#print(cz)

return cz

def Run():

cz = ReadInput()

if len(cz) == 0: print(0); return

print(1)

# print(cz)

cle = Cycle(cz)

print('Found a Cycle:', cle)

# shorted cycle in connected zone cz

def Cycle(cz): # backtracking

global visit, v

if len(cz) < 3: return []

if len(cz) == 3: cz + [cz[0]]

visit = bytearray([0]\*len(d))

v = [0]\*len(cz) # path

x = cz[0]

k = 0 # bước đi

v[k] = x # xuat phat tu dinh s

visit[x] = 1 # da tham dinh s

while True:

vv = Goal(cz)

if len(vv) > 3: return vv # den dich

if k < 0: return []

z = Next(cz, v[k]) # v[k] -> z ?

if z > 0: # tien

k += 1

v[k], visit[z] = z, 1

else: k -= 1 # lui

# Tìm đường đi từ x -> z

def Next(cz, x):

for z in cz:

if x != z and not visit[z] and a[x][z]:

return z

return -1

"""

Đã đến đích ?

xét các đỉnh đã thăm trong path v

Gọi z là đỉng cuối trong số các đỉnh trên

và y là đỉnh sát trước z

Nếu có cạnh (x,z), x trước y tức là có

chu trình

"""

def Goal(cz):

# xét các đỉnh x > 0 trong path v

vv = []

for x in v:

if x > 0: vv.append(x)

if len(vv) < 3: return []

# len(vv) >= 3: vv = [...z]

z = vv[-1]

for x in vv[:-2]: # cạnh (x,z) tạo chu trình

if a[x][z]:

i = vv.index(x)

return vv[i:]+[x]

return []

# APPLICATION

Run()

print(" T h e E n d")

#### Output

1

Found a Cycle: [3, 4, 8, 6, 3]

T h e E n d

## Shortest Cycle

*Cho đồ thị vô hướng G = (V, E) với n đỉnh V = {1..n} và m cạnh (x, y) nối đỉnh x với đỉnh y. Hãy tìm một chu trình ngắn nhất trong G.*

Ví dụ

|  |  |  |
| --- | --- | --- |
| graph.inp | Output | A diagram of a network  Description automatically generated |
| 10 10  9 1  2 5  4 7  8 6  5 1  4 3  9 10  10 2  4 8  3 6 | 3 4 8 6 3 |

#### Algorithm

1. Dùng kỹ thuật Find-Union tìm và đánh dấu các TPLT của đồ thị

2. Với mỗi TPLT cz gọi thuật toán quay lui Cycle(cz) tìm chu trình ngắn nhất trong cz. Ghi nhận mincycle của các ứng viên

#### Độ phức tạp

, n là số đỉnh, k là số TPLT của đồ thị

#### Program

# Shortest Cycle

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

""" Đọc dữ liệu vào ma trận kề a

Dùng kỹ thuật Find-Union để xác định TPLT chứa cycle

n: số đỉnh

cd: TPLT chứa cycle

"""

def ReadInput():

global a, d, cd

cd = set({}) # tập các TPLT chứa cicle

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

a = [[0]\*(n+1) for i in range(n+1)] # matrận kề

d = list(range(n+1)) # mảng tham chiếu

for i in range(m): # cạnh (x,y)

x, y = map(int,f.readline().strip().split())

if x == y: continue

a[x][y] = a[y][x] = 1

if Union(x,y) == 0: # gặp cycle

cd.add(Find(x)) # Ghi nhận TPLT chứa cycle

# đặt lại trỏ trực tiếp

for i in range(n+1): d[i] = Find(i)

def Run():

ReadInput()

if cd == []:

print('No Cycles')

return

#print(cd); print(d); print(len(d)); Go()

# Tách các TPLT vào cz

mincycle = list(range(len(d)))

count = 0

for i in cd:

cz = [] # TPLT in vertex i

for j in range(i, len(d)):

if d[j] == i: cz.append(j)

cle = Cycle(cz) # Find a cycle cle

if len(cle) == 0: continue

count += 1

print('Cycle No', count, cle)

if len(cle) == 3:

mincycle = cle

break

if len(cle) < len(mincycle): mincycle = cle

print('Shorted Cycle:', mincycle)

# shorted cycle in connected zone cz

def Cycle(cz): # backtracking

global visit, v

if len(cz) < 3: return []

if len(cz) == 3: cz + [cz[0]]

visit = bytearray([0]\*len(d))

v = [0]\*len(cz) # path

x = cz[0]

k = 0 # bước đi

v[k] = x # xuat phat tu dinh s

visit[x] = 1 # da tham dinh s

while True:

vv = Goal(cz)

if len(vv) > 3: return vv # den dich

if k < 0: return []

z = Next(cz, v[k]) # v[k] -> z ?

if z > 0: # tien

k += 1

v[k], visit[z] = z, 1

else: k -= 1 # lui

# Tìm đường đi từ x -> z

def Next(cz, x):

for z in cz:

if x != z and not visit[z] and a[x][z]:

return z

return -1

"""

Đã đến đích ?

xét các đỉnh đã thăm trong path v

Gọi z là đỉng cuối trong số các đỉnh trên

và z1 là đỉnh sát trước z

Nếu có cạnh (x,z), x trước z1 tức là có

chu trình

"""

def Goal(cz):

# xét các đỉnh x > 0 trong path v

vv = []

for x in v:

if x > 0: vv.append(x)

if len(vv) < 3: return []

# len(vv) >= 3: vv = [...z]

z = vv[-1]

for x in vv[:-2]: # cạnh (x,z) tạo chu trình

if a[x][z]:

i = vv.index(x)

return vv[i:]+[x]

return []

# APPLICATION

Run()

print(" T h e E n d")

#### Output

Cycle No 1 [1, 5, 2, 10, 9, 1]

Cycle No 2 [3, 4, 8, 6, 3]

Shorted Cycle: [3, 4, 8, 6, 3]

T h e E n d

## Net

*Việt Nam*

*Một phòng học có n máy tính mang mã số từ 1 đến n, trong đó có m cặp máy tính được nối với nhau. Hãy cho biết cần nối thêm ít nhất bao nhiêu cặp máy tính để tạo thành một mạng theo nghĩa hai máy tính bất kỳ có thể trực tiếp gửi thông điệp cho nhau hoặc thông qua một số máy tính trung gian.*

|  |  |  |
| --- | --- | --- |
| Net.inp | Output | *Input: 8 máy tính, 5 cặp máy được nối với nhau.*  *Output: Cần nối thêm 2 cặp máy 1-3 và 1-6.*  A diagram of numbers and circles  Description automatically generated |
| 8 5  2 5  3 7  8 6  5 1  4 3 | 2  1 3  1 6 |

#### Algorithm

Quan sát ví dụ trong đề bài ta dễ dàng phát hiện ra thuật toán:

*Bước 1.* Gọi thuật toán Find–Union để cập nhật mảng tham chiếu d và tính số thành phần liên thông k

*Bước 2.* Quyết định:

* Nếu k = 1, tức là đồ thị là liên thông thì không cần nối thêm cặp máy tính nào.
* Nếu k > 1 thì nối thêm k−1 cặp máy tính (1, i) trong đó i là "nhóm trưởng". Để ý rằng 1 là số hiệu nhỏ nhất nên 1 luôn luôn là nhóm trưởng, do đó ta chỉ cần nối máy 1 với các máy là nhóm trưởng còn lại.

#### Program

# Net

FN = 'NET.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

def Net():

global d

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

d = list(range(n+1))

for i in range(m):

x, y = map(int,f.readline().strip().split())

Union(x, y) # cạnh (v[j], v[j+1])

# danh sách các nhóm trưởng trừ máy số 1

head = [i for i in range(2, n+1) if d[i] == i]

print('Cần nối thêm', len(head), 'cặp máy:')

for v in head: print('1 -',v)

# APPLICATION

Net()

print(" T h e E n d")

#### Output

Cần nối thêm 2 cặp máy:

1 - 3

1 - 6

T h e E n d

## Water Valves

*ACM*

*Có N thùng nước được đánh số từ 1 đến N, giữa 2 thùng bất kỳ đều có một ống nối, trên ống có một van có thể đóng (khóa: không cho nước lưu thông trong ống đó) hoặc mở cho nước chảy trong ống. Ở trạng thái ban đầu tất cả các van đều đóng.*

*Bạn được cho một số yêu cầu thuộc 2 dạng với ý nghĩa như sau:*

*Dạng X Y 1: bạn cần mở van trên ống nối giữa 2 thùng X và Y.*

*Dạng X Y 2: bạn cần cho biết với trạng thái các van đang mở/đóng như hiện tại thì 2 thùng X và Y có thuộc cùng một nhóm bình thông nhau hay không, nghĩa là nước từ bình này có thể chảy đến được bình kia qua một số ống có van đang mở.*

*Input text file:*

* *Dòng đầu tiên ghi số thùng nước N và số yêu cầu P,* 1 ≤ *N* ≤ 10000, 1 ≤ *P* ≤ 50000.
* *Trong P dòng tiếp theo, mỗi dòng ghi ba số nguyên dương X, Y, Z với ý nghĩa có yêu cầu loại Z (Z = 1 hoặc Z = 2) với 2 thùng X và Y.*

*Output:*

* *Với mỗi yêu cầu dạng X Y 2 (Z = 2) bạn cần ghi trên một dòng một trong 2 kết quả 0 hoặc 1 với ý nghĩa như sau:*
  + *0: hai thùng X và Y thông nhau;*
  + *1: hai thùng X và Y không thông nhau.*

*Ví dụ*

|  |  |  |
| --- | --- | --- |
| VALVES.INP | Output |  |
| 7 9 |  | *7 thùng, 9 yêu cầu* |
| 1 2 2 | 0 | *Thùng 1 và 2 không liên thông* |
| 1 2 1 |  |  |
| 3 7 2 | 0 | *Thùng 3 và 7 không liên thông* |
| 2 3 1 |  |  |
| 1 3 2 | 1 | *Thùng 1 và 3 liên thông* |
| 2 4 2 | 0 | *Thùng 2 và 4 không liên thông* |
| 1 4 1 |  |  |
| 3 4 2 | 1 | *Thùng 3 và 4 liên thông* |
| 1 7 2 | 0 | *Thùng 1 và 7 không liên thông* |

#### Algoithm

Bài này khá đơn giản. Hàm Van dưới đây đọc dần từng dòng lệnh x y z và xét:

* Nếu z = 1: thực hiện Union(x,y) để gộp TPLT chứa thùng x với TPLT chứa thùng y cho kết quả là TPLT chứa đồng thời hai thùng x và y.
* Nếu z = 2: Xét
* Nếu Find(x) = Find(y) tức là hai thùng x và y thuộc cùng một TPLT nên x và y *thông nhau*;
* Nếu Find(x) ≠ Find(y) tức là hai thùng x và y thuộc hai TPLT khác nhau nên x và y *không thông nhau.*

Chú ý: Khi cài đặt chúng ta thường quên gọi thủ tục khởi trị mảng tham chiếu d!!!

#### Program

# water valves

FN = 'VALVES.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

def Valves():

global d

with open(FN) as f:

v = list(map(int,f.read().split()))

n, m = v[0], v[1] # n đỉnh, m cạnh

d = list(range(n+1))

j = -1

for i in range(m):

j += 3

x, y, z = v[j], v[j+1], v[j+2]

if z == 1: Union(x,y)

else: # z == 2 ask

if Find(x) == Find(y): print(1)

else: print(0)

# APPLICATION

Valves()

print(" T h e E n d")

## Bridges

*Một cạnh (x,y) trong đồ thị vô hướng G được gọi là cầu hoặc cạnh trọng yếu nếu xóa cạnh đó đi (giữ lại các đỉnh x và y ở hai đầu), sẽ làm tăng số thành phần liên thông của đồ thị. Hãy liệt kê các cầu của đồ thị n đỉnh và m cạnh.*

Ví dụ

|  |  |  |
| --- | --- | --- |
| GRAPH.INP | Output | *Input: Đồ thị có 8 đỉnh, 7 cạnh.*  *Output: Đồ thị có 3 cầu* (1,5), (2,5) và (3,7)  (**tô đậm**)  A diagram of a network  Description automatically generated |
| 8 7  3 7  1 5  3 4  2 5  3 8  4 6  6 8 | 3  1 5  2 5  3 7 |

#### Algorithm

Nhận xét 1: *Mọi cầu phải là cạnh khung, nghĩa là phải thuộc một cây hoặc rừng khung.*

Thật vậy, nếu cạnh (x,y) không là cạnh khung thì khi xét cạnh đó trong thủ tục tìm cây khung ta phải có Union(x,y) = 0. Thế tức là đồ thị có chu trình chứa cạnh (x,y) này. Khi bỏ cạnh (x,y) trong chu trình ta thấy các đỉnh nằm trên chu trình này vẫn liên thông với nhau, do đó số thành phần liên thông là không đổi. Điều này cho thấy (x,y) không thể là cầu.

Chú ý rằng mệnh đề đảo của mệnh đề trên không đúng. Cạnh (3,4) của hình trong ví dụ có thể nằm trong rừng khung nhưng (3,4) *không phải là cầu*.

Ta gọi *bậc* của đỉnh x trong đồ thị vô hướng là số cạnh đi ra (hoặc đi đến) đỉnh đó.

Nhận xét 2: *Nếu một cạnh có một đầu bậc 1 thì cạnh đó là cầu.*

Thạt vậy, giả sử cạnh (x,y) có bậc của đỉnh x là 1 thì khi xóa cạnh đó ta biến đỉnh x thành đỉnh cô lập tức là x được tách thành một TPLT, do đó số TPLT được tăng lên.

Hai nhận xét trên cho phép ta chỉ kiểm tra những cạnh khung để phát hiện cầu.

* *Bước 1.* Đọc dữ liệu, dùng kỹ thuật Find-Union ghi nhận các giá trị sau đây:
* số mảnh liên thông
* mảng chứa các cạnh khung.
* Tính bậc của các đỉnh

Độ phức tạp: O(nm)

* *Bước 2.* Duyệt các cạnh khung (x,y).
* Nếu bậc của x hoặc y là 1 thì đưa vào cầu
* Nếu thử bỏ cạnh (x,y) mà số thành phần liên thông tăng lên thì (x,y) là cầu,

Độ phức tạp:

Hàm FU(c) vận dụng kỹ thuật Find−Union để xác định số thành phần liên thông của đồ thị khuyết cạnh thứ i.

#### Program

# Bridges

FN = 'GRAPH.INP'

EDGE = 1 # cạnh của đồ thị

SP = 2 # cạnh khung

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

"""

Đọc dữ liệu tạo

n: số đỉnh

dict cạnh e {c(x,y):mark}

d: mảng tham chiếu

k: số TPLT

"""

def Input():

global n, d, e, k, deg

e = {} # cạnh:1; cạnh khung: 2; xóa cạnh: 0; cầu: 3

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

# n đỉnh, m cạnh

d = list(range(n+1)) # mảng tham ciếu (trỏ ngượ)

deg = [0]\*(n+1) # bậc của đỉnh

k = n

for i in range(m):

x, y = map(int,f.readline().strip().split())

if x == y: continue

if x > y: x, y = y, x

deg[x] += 1; deg[y] += 1

c = (x,y)

# Ghi nhận c là cạnh EDGE hoặc là cạnh khung SP

if Union(x,y): e[c] = SP; k -= 1

else: e[c] = EDGE

for i in range(1, n+1): d[i] = Find(i) # Đặt lại trỏ trực tiếp

def Bridges():

global d

Input()

#print(n, 'đỉnh')

print(k, 'TPLT')

b = [] # Bridges

for c in e:

if deg[c[0]] == 1 or deg[c[1]] == 1:

b.append(c)

continue

# xóa cạnh khung c giữ lại đỉnh

if e[c] == SP:

if FU(c): b.append(c)

print('Result:',len(b), '\n', b)

"""

Thực hiện Find-Union sau khi xóa cạnh khung c

"""

def FU(c):

global d

d = list(range(n+1))

t = n

for xy in e:

if xy != c:

t -= Union(xy[0], xy[1])

return t > k

# APPLICATION

Bridges()

print(" T h e E n d")

#### Output

2 TPLT

Result: 3

[(3, 7), (1, 5), (2, 5)]

T h e E n d

## Intersection Of Spanning Trees

*Ta quy ước gọi khung là cây hoặc rừng khung của một đồ thị. Đồ thị có thể có một hoặc nhiều khung. Hãy liệt kê giao của mọi khung trong đồ thị vô hướng n đỉnh, m cạnh, tức là liệt kê các cạnh có mặt trong mọi khung của đồ thị.*

Gợi ý: *Giao của khung = Tập các cầu.*

## Joint Vertex

*Đỉnh khớp của một đồ thị vô hướng là đỉnh thỏa tính chất sau: nếu ta xóa đỉnh đó và các cạnh liên thông thuộc với đỉnh đó sẽ làm tăng số thành phần liên thông của đồ thị. Hãy liệt kê các đỉnh khớp trong đồ thị vô hướng n đỉnh, m cạnh.*

*Chú ý:* Bạn cần phân biệt các thuật ngữ sau:

1. *Xóa cạnh* x-y được hiểu là loại bỏ đường nối giữa hai đỉnh x và y nhưng *giữ lại* hai đỉnh đó. Về mặt trực quan ta hiểu như sau:

Trước khi xóa cạnh x−y: (x)−−(y)

Sau khi xóa cạnh x-y:(x) (y)

1. Xóa đỉnh x được hiểu là loại bỏ đỉnh x và mọi cạnh liên thông thuộc đỉnh đó. Về mặt trực quan ta hiểu như sau:

Trước khi xóa đỉnh x: (z)−−(x)−−(y)

Sau khi xóa đỉnh x:(z) (y)

#### Ví dụ

|  |  |  |
| --- | --- | --- |
| Input | Output |  |
| 8 7  1 5  2 5  3 7  3 4  3 8  4 6  6 8 | 2  3  5 | *Input: Đồ thị có 8 đỉnh, 7 cạnh.*  *Output: Đồ thị có 2 đỉnh khớp: 3 và 5. (****tô đậm****)*  A diagram of a network  Description automatically generated |

#### Algorithm

Nhận xét: Ta gọi bậc của đỉnh v trong đồ thị vô hướng là số cạnh đi ra (hoặc đi đến) đỉnh đó. Dễ thấy là nếu đỉnh v có bậc < 2 thì v không thể là đỉnh khớp.

* Đọc input file lưu m cạnh vào set e đồng thời dùng Find-Union tính số mảnh liên thông k hiện có của đồ thị và bậc của các đỉnh. Độ phức tạp:
* Duyệt lần lượt n đỉnh v có bậc ≥ 2: gọi hàm FU(v) tính lại số mảnh liên thông của đồ thị n−1 đỉnh sau khi bỏ đỉnh v và các cạnh liên thuộc với đỉnh đó.

Nếu FU(v) > k thì v là đỉnh khớp. Ghi nhận đỉnh này.

Cạnh liên thuộc đỉnh v là cạnh có đỉnh đầu hoặc đỉnh cuối là v.

Độ phức tạp:

#### Program

# Joint Vertex

FN = 'GRAPH.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

"""

Đọc dữ liệu tạo

n: số đỉnh

set e: tập cạnh

d: mảng tham chiếu

k: số TPLT

deg: bậc của đỉnh

"""

def Input():

global n, e, d, k, deg

e = set({})

with open(FN) as f:

n, m = map(int,f.readline().strip().split())

d = list(range(n+1))

deg = [0]\*(n+1) # bậc của đỉnh

k = n

print(n, 'đỉnh', m, 'cạnh')

#print(v); Go()

for i in range(m):

x, y = map(int,f.readline().strip().split())

if x == y: continue

if x > y: x, y = y, x

deg[x] += 1; deg[y] += 1

k -= Union(x,y)

e.add((x,y))

# Đặt lại trỏ trực tiếp

for i in range(1, n+1): d[i] = Find(i)

def JV(): # Join Vertex

Input()

# print(e)

print('TPLT:', k)

jv = [] # các đỉnh khớp

for v in range(1,n+1):

if deg[v] < 2: continue

if FU(v): jv.append(v)

print(len(jv), 'đỉnh khớp:')

for v in jv: print(v)

"""

Find-Union of vertex v:

Thử bỏ đỉnh v tình lại STPLT

xét các cạnh (x,y) x != v and y != v

"""

def FU(v):

global d

d = list(range(n+1))

k1 = n-1 # số TPLT sau khi bổ đỉnh v

for c in e:

if c[0]==v or c[1]==v: continue

k1 -= Union(c[0], c[1])

#print('Vertex',v,' k1 =',k1)

return k1 > k

# APPLICATION

JV()

print(" T h e E n d")

#### Output

8 đỉnh 7 cạnh

TPLT: 2

2 đỉnh khớp:

3

5

T h e E n d

## Connections

*Một phòng học có n máy tính mang mã số từ 1 đến n, trong đó có m cặp máy tính được nối với nhau. Hãy cho biết cần gỡ đi nhiều nhất bao nhiêu đoạn cáp nối các cặp máy tính và cần nối thêm ít nhất là bao nhiêu cặp máy để tạo thành một mạng theo nghĩa hai máy tính bất kỳ có thể trực tiếp gửi thông điệp cho nhau hoặc thông qua một số máy tính trung gian.*

|  |  |  |
| --- | --- | --- |
| GRAPH.INP | Output | *Input: 8 máy tính, trong đó có 10 cặp máy được nối với nhau.*  *Output: Cần gỡ 4 đoạn cáp 1-5, 4-8, 6-8 và 7-8 (cạnh …);*  *Nối thêm 1 cặp máy 1-3 (cạnh tô đậm).*  *Chú ý: Lúc đầu Input không có cạnh 1-3.*  A diagram of a network  Description automatically generated |
| 8 10  2 5  1 2  1 5  3 7  3 8  4 6  4 7  4 8  6 8  7 8 | -4  1 5  4 8  6 8  7 8  +1  1 3 |

#### Algorithm

Quan sát hình vẽ trong ví dụ ta dễ dàng phát hiện ra thuật toán:

* *Bước 1.* Gọi hàm FU (Find-Union) để cập nhật mảng tham chiếu d và tính số thành phần liên thông k và đánh dấu các cạnh khung trong dict e {(x,y):mark}.
* *Bước 2.* Quyết định:
* Số cạnh khung ck = n − k
* Số cạnh cần gỡ = m − ck (chỉ giữ lại các cạnh khung)
* Số cạnh cần nối thêm = k − 1

#### Program

# Connectivity

FN = 'GRAPH.INP'

EDGE = 1 # cạnh thường

SP = 2 # cạnh khung

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

"""

Đọc dữ liệu tạo

n: số đỉnh

dict e{(x,y):mark} từ điển cạnh

d: mảng tham chiếu

k: số TPLT

"""

def Input():

global n, e, d, k

e = {}

with open(FN) as f:

data = list(map(int,f.read().split()))

n, m = data[0], data[1] # n đỉnh, m cạnh

d = list(range(n+1))

k = n

print('Số đỉnh:', n, ' Số cạnh:',m)

j = 0

for i in range(m):

j += 2

x,y = data[j], data[j+1]

if x == y: continue

c = (x,y) if x < y else (y,x) # quy định x < y

if not c in e: e[c] = SP

if Union(c[0], c[1]): k -= 1

else: e[c] = EDGE

# Update trỏ trực tiếp

for i in range(1,n): d[i] = Find(i)

print(d)

for c in e: print(c, ':', e[c])

print('TPLT:',k)

def Connect():

Input()

m = len(e)

ck = n - k

print(ck-m)

for c in e:

if e[c] == EDGE: print(c)

print('+',k-1)

for i in range(2,n):

if d[i] == i: print(1, '-',i)

# APPLICATION

Connect()

print(" T h e E n d")

#### Output

Số đỉnh: 8 Số cạnh: 10

[0, 1, 1, 3, 3, 1, 3, 3, 3]

(2, 5) : 2

(1, 2) : 2

(1, 5) : 1

(3, 7) : 2

(3, 8) : 2

(4, 6) : 2

(4, 7) : 2

(4, 8) : 1

(6, 8) : 1

(7, 8) : 1

TPLT: 2

-4

(1, 5)

(4, 8)

(6, 8)

(7, 8)

+ 1

1 - 3

T h e E n d

## IT Teams

*Có n bạn trẻ đến dự lễ hội Tin học. Khi vào cửa, mỗi bạn được một robot cấp một mã gồm 6 chữ cái in hoa 'A'…'Z'. Tại hội trường, Ban Tổ chức yêu cầu các bạn đứng theo từng đội để tham gia đấu trí. Hai người có mã giống nhau dù chỉ một chữ cái sẽ vào cùng một đội. Ví dụ, bạn trẻ 'ABCDAB' và bạn trẻ 'DXEZEV' sẽ cùng vào một đội. Nếu A và B cùng đội, B và C cùng đội thì đương nhiên là ba bạn đó sẽ cùng đội. Hãy hiển thị trên màn hình số đội và liệt kê số hiệu của từng người trong đội tính từ 1 đến n.*

*Input text file "Team.inp": Dòng đầu tiên số n; max(n) = 2000;*

*Dòng thứ i trong số n dòng tiếp theo: mã của bạn thứ i*

*Ví dụ*

|  |  |  |
| --- | --- | --- |
| Team.inp | Output | *8 bạn:*  *Bạn số 1 có mã* ***ABBBAA***  ***…***  *Bạn số 8 có mã* ***VIETVI***  ***3*** *đội:*  *Đội 1:* ***1 4 5***  *Đội 2:* ***2 6***  *Đội 3:* ***3 7 8*** |
| 8  ABBBAA  CDCDCD  EEEEEE  FAAAAF  GGGFFF  DXXXXX  VTVTVT  VIETVI | 3  1 4 5  2 6  3 7 8 |

#### Algorithm

Ta mường tượng mỗi bạn trẻ là một đỉnh của đồ thị, hai bạn x và y có tập các chữ cái trong mã giao nhau (tức là chung nhau ít nhất một chữ cái) thì tạo thành một cạnh x−y của đồ thị. Khi đó số nhóm chính là số TPLT của đồ thị.

*Bước 1. Đ*ọc dữ liệu và tạo ra các tập chữ cái trong mã của các bạn trẻ. Kiểm tra giao của các tập để xác định điều kiện thực hiện hàm Union(i,j) rồi hợp nhất các nhóm bạn trẻ thành các TPLT.

*Bước 2.* Duyệt lại mảng liên kết d để hiển thị các thành phần liên thông, tức là các thành viên trong mỗi đội.

Độ phức tạp: Với mỗi bạn i = 1..n ta phải duyệt lại các bạn đọc trước i do đó ta cần thời gian n2. Thao tác Union cần thời gian n, vậy tổng cộng độ phức tạp là n3.

#### Program

# IT Teams

FN = 'TEAM.INP'

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# nhóm trưởng của nhóm chứa x

def Find(x):

while d[x] != x: x = d[x]

return x

# Gộp nhóm chứa x với nhóm chứa y

def Union(x,y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

"""

Đọc dữ liệu

n: số đỉnh

code: danh sách tập các mã số thu gọn

d: mảng tham chiếu

k: số TPLT

"""

def Team():

global n, d, k

with open(FN) as f:

data = f.read().strip().split()

#print(data)

n = int(data[0]) # n đỉnh

d = list(range(n+1))

k = n # số TPLT

#print('Số đỉnh:', n)

code = list(map(set, data))

print(code); Go()

for i in range(2,n+1):

for j in range(1,i):

if code[i].isdisjoint(code[j]): continue

k -= Union(i,j)

for i in range(1, n+1): d[i] = Find(i) # Trỏ trực tiếp

#print(d); print(k)

# result

print(k)

for i in range(1,n+1):

if d[i] == i:

for j in range(i,n+1):

if d[j] == i: print(j, end = ' ')

print()

# APPLICATION

Team()

print(" T h e E n d")

#### Output

3

1 4 5

2 6

3 7 8

T h e E n d

## Ants

*Bi là cậu bé rất thích nuôi kiến vì Bi cho rằng kiến rất giỏi tìm đường đi ngắn nhất trong mê cung. Bi dùng đất nặn để xây dựng một mê cung hoành tráng gồm n lòng chảo và m đường hào nối các lòng chảo với nhau. Sau khi thả kiến vào một lòng chảo Bi phát hiện ra rằng kiến không bao giờ bò đến một số lòng chảo. Bạn hãy cho biết Bi cần làm thêm ít nhất là bao nhiêu đường hào để kiến có thể xuất phát từ một lòng chảo bất kỳ tìm được đến một lòng chảo bất kỳ, nơi Bi đã đặt sẵn cục đường phèn.*

*Input text file "ANT.INP":*

*Dòng đầu tiên hai số nguyên dương n và m; max(n) = 2000; max(m) = 5000;*

*Dòng thứ i trong số m dòng tiếp theo: hai số nguyên dương x và y thể hiện một đường hào nối trực tiếp lòng chảo x với lòng chảo y.*

*Output:*

*Dòng đầu tiên: số nguyên k cho biết số đường hào tối thiểu cần xây thêm.*

*Dòng thứ i trong số k dòng tiếp theo: hai số nguyên dương u và v thể hiện một đường hào cần làm thêm để nối lòng chảo u với lòng chảo v.*

Ví dụ

|  |  |
| --- | --- |
| ANT.INP | Output |
| 8 5  2 5  3 7  8 6  5 1  4 3 | 2  1 3  1 6 |

#### Algorithm

Xem bài Net

## Subgraph

Cho đồ thị vô hướng G gồm N đỉnh có số hiệu từ 1 đến N và M cạnh. Giữa hai đỉnh khác nhau của G có không quá một cạnh nối hai đỉnh đó. Cho số hiệu của hai đỉnh S và E trong đồ thị, 1 ≤ S ≤ E ≤ N. Xét đồ thị con H(S,E) của G gồm các đỉnh có mã số từ S đến E và các cạnh (x,y) nối hai đỉnh nằm trọn trong khoảng từ S đến E: S ≤ x, y ≤ E.

Với mỗi bộ ba giá trị S, E và V, S ≤ V ≤ E cần hiển thị vùng liên thông chứa đỉnh V của đồ thị con H(S, E).

Input: text file GRAPH.INP

Dòng đầu tiên: 2 số N M được ghi cách nhau.

Dòng tiếp theo: M giá trị x[1] … x[M] được ghi cách nhau.

Dòng tiếp theo: M giá trị y[1] … y[M] được ghi cách nhau.

(x[j], y[j]) là cạnh thứ j của đồ thị liên thông G, 1 ≤ j ≤ M.

Dòng tiếp theo: số test Q

Q dòng tiếp theo: mỗi dòng ba số S, E, V, 1 ≤ S ≤ V ≤ E ≤ N được ghi cách nhau.

Giới hạn: 1 ≤ N ≤ 200000; 1 ≤ M ≤ 400000; 1 ≤ Q ≤ 400000.

#### Ví dụ

|  |  |  |
| --- | --- | --- |
| GRAPH.INP | *Đồ thị liên thông có N = 8 đỉnh,*  *M = 7 cạnh: (1,6), (1,5), (4,1), (4,8),*  *(3,5), (7,3), (5,2).*  *Q = 2 test*  *Test 1. S = 1, E = 4, V = 4: Hiển thị: 1 4.*  *Test 2. S = 3, E = 8, V = 7: Hiẻn thị: 3 5 7.* | A diagram of a network  Description automatically generated |
| 8 7  1 1 4 4 3 7 5  6 5 1 8 5 3 2  2  1 4 4  3 8 7 |

#### 

|  |  |
| --- | --- |
| A diagram of circles with numbers  Description automatically generated | A diagram of a network  Description automatically generated |
| Test 1: S = 1, E = 4, V = 4  H(S, E) có ba miền liên thông.  *Đỉnh V = 4 trong miền liên thông 1 4.* | Test 2: *S = 3, E = 8, V= 7*  *H(S, E) có ba miền liên thông*  *Đỉnh V = 7 trong miền liên thông*  *3 5 7* |

#### Algorithm

Thực hiện Find-Union để gán trị cho mảng tham chiếu d theo các cạnh (*x,y*) nằm trọn trong đồ thị con, Subgraph(S,E) nghĩa là thỏa mãn điều kiện S ≤ *x, y* ≤ E.

Để tránh việc duyệt mọi cạnh của đồ thị ta tổ chức một từ điển vt {x:s} trong đó x là đỉnh của đồ thị, s là tập các đỉnh y liên thuộc đỉnh x, y > x.

Ví dụ trong đề bài cho ta:

vt[1] = {4, 5, 6}

vt[2] = {5}

vt[3] = {5, 7}

vt[4] = {8}

vt[5] = vt[6] = vt[7] = vt[8] = {}

Hàm Less(s, e) cho ra tập các phần tử y trong tập s thỏa điều kiện lọc y ≤ e.

#### Độ phức tạp

* Đọc dữ liệu: O(NM)
* Tính ConnectedZone(s,e,v):

#### Program

# SUBGRAPH

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def Find(x):

while d[x] != x: x = d[x]

return x

def Union(x, y):

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

# lọc các phần tử x trong set s thỏa x <= e

def Less(s, e): return set(filter(lambda y: y <= e, s))

# TPLT chứa v của subgraph(s..e)

def ConnectedZone(s, e, v):

global d

d = list(range(n+1))

r = range(s,e+1)

for x in r:

for y in Less(vt[x], e): Union(x,y)

for i in r: d[i] = Find(i) # đặt lại trỏ trực tiêp

return set(j for j in r if d[j] == d[v])

def Run():

global n, m, vt

"""

Đọc dữ liệu

n đỉnh, m cạnh

tạo từ điển vt{đỉnh x:{các đỉnh y > x}}

"""

with open('graph.inp') as f:

n, m = map(int, f.readline().strip().split())

print(n,'đỉnh', m, 'cạnh')

sx = list(map(int, f.readline().strip().split()))

sy = list(map(int, f.readline().strip().split()))

vt = {i: set({}) for i in range(1, n+1)}

for i in range(m): # đọc các cạnh

x, y = sx[i], sy[i]

if x == y: continue

if x < y: vt[x].add(y)

else: vt[y].add(x)

# print(vt)

q = int(f.readline()) # số test

print(q, 'test')

for t in range(1, q+1):

print(' Test', t, end=': ')

s, e, v = map(int, f.readline().strip().split())

print(s, e, v)

print('Result:', ConnectedZone(s, e , v))

# application

Run()

#### Output

8 7

2 test

Test 1: 1 4 4

Result: {1, 4}

Test 2: 3 8 7

Result: {3, 5, 7}

## Werewolf

#### IOI Japan, 2018

Quận Ibaraki ở Nhật Bản có N thành phố và M con đường. Các thành phố được đánh số từ 0 đến N−1 theo thứ tự tăng dần về số lượng dân trong thành phố. Mỗi con đường nối một cặp hai thành phố khác nhau, và có thể di chuyển theo cả hai hướng. Bạn có thể sử dụng một hoặc nhiều con đường này để di chuyển từ một thành phố bất kì đến một thành phố bất kì khác.

Bạn lên kế hoạch gồm Q chuyến đi, được đánh số từ 0 đến Q−1. Chuyến đi thứ i (0 ≤ i ≤ Q−1) sẽ di chuyển từ thành phố Si đến thành phố Ei.

Bạn là người sói. Bạn có hai hình dáng: **Hình dáng người** và **hình dáng sói**. Tại thời điểm bắt đầu của mỗi chuyến đi, bạn ở hình dáng người. Kết thúc chuyến đi, bạn bắt buộc phải ở hình dáng sói. Trong chuyến đi, bạn phải **chuyển đổi** (chuyển từ hình dáng người sang hình dáng sói) đúng một lần. Bạn chỉ có thể chuyển đổi khi bạn đang ở một thành phố nào đó (kể cả Si hoặc Ei).

Cuộc sống của người sói không dễ dàng. Bạn phải tránh ở hình dáng người khi ở các thành phố ít dân và tránh ở hình dáng sói khi ở các thành phố đông dân. Với mỗi chuyến đi i (0 ≤ i ≤ Q−1), có hai ngưỡng Li và Ri (0 ≤ Li ≤ Ri ≤ N−1 ) chỉ ra các thành phố nào bạn phải tránh. Cụ thể là, bạn không được ở hình dáng người khi ở các thành phố 0, 1, …, Li−1, và không được ở hình dáng sói khi ở các thành phố Ri+1, Ri+2, …, N−1. Điều đó có nghĩa là trong chuyến đi thứ i, bạn chỉ được chuyển đổi hình dáng ở một trong các thành phố Li, Li+1,…,Ri.

Nhiệm vụ của bạn là xác định xem đối với mỗi chuyến đi có tồn tại hay không cách di chuyển từ thành phố Si đến Ei thỏa mãn tất cả các điều kiện ở trên. Chuyến đi của bạn có thế có độ dài bất kì.

#### Understanding

Ta tinh giản đề như sau:

Cho đồ thị G vô hướng, liên thông gồm N đỉnh và M cạnh. Các đỉnh được mã số từ 0 đến N−1. Cho hai đỉnh giới hạn L và R, 0 ≤ L ≤ R ≤ N.

Xét hai đồ thị con H và W của G.

* H gồm các đỉnh từ L đến N−1 và các cạnh chứa các đỉnh đó.
* W gồm các đỉnh từ 0 đến R và các cạnh chứa các đỉnh đó.

Cho hai đỉnh S ∈ H và E ∈ W.

Hãy cho biết có tồn tại hay không một đường đi gồm hai đoạn h và w thỏa mãn các điều kiện sau:

* h là đường đi trong đồ thị con H xuất phát từ S kết thúc tại đỉnh V.
* w là đường đi trong đồ thị con W xuất phát từ V kết thúc tại E?

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | ... | L - 1 | L | L + 1 | ... | R - 1 | R | R + 1 | ... | N - 1 |
|  | | | | | S ∈ H người | | | | | | | |
| E ∈ W sói | | | | | | | | | |  | | |
|  |  |  |  |  | chuyển đổi người - sói | | | | |  |  |  |

#### Algorutem

Ý tưởng của thuật toán khá đơn giản: Ta cần tìm một đường đi

đáp ứng được các tiêu chí sau:

* : Xuất phát từ đỉnh S trong đồ thị con H, kết thúc tại đỉnh E trong đồ thị con W.
* : đoạn đường này nằm trọn trong H.
* : đoạn đường này nằm trọn trong W.

Từ đây suy ra rằng đoạn đường phải nằm trong phần giao của hai đồ thị con H và W, và đây cũng chính là nơi chuyển đổi từ lốt người sang lốt sói.

Bài toán Werwolf sẽ có nghiệm khi và chỉ khi giao của hai tập đỉnh H(S) và W(E) không rỗng.

Thật vậy, gọi V là điểm trong H(S) ∩ W(E). Do V ∈ H(S), H(S) là miền liên thông trong đồ thị con H và chứa đỉnh S nên từ S có thể di chuyển trong H để đến V, nghĩa là có đường đi trong H từ S đến V. Ta kí hiệu đường đi này là h: S → V. Tương tự, V ∈ W(E) khi và chỉ khi có đường đi trong đồ thị con W từ V đến E, ký hiệu là V → E. Từ đó suy ra là có đường đi hai chặng từ S đến E như sau:

S → V → E. Cụ thể là

Từ S trong H bạn di chuyển đến V trong lốt Người: (HS) → (HV).

Tại V bạn đổi sang lốt sói: (HV)/(WV).

Rồi bạn di chuyển tiếp trong lốt sói từ V đến E: (WV)→ (WE).

Lưu ý rằng do V là giao của H(S) và W(E) nên L ≤ V ≤ R, nghĩa là tại V bạn có thể chuyển đổi lốt từ người sang lốt sói. Vì lý do này nên ta gọi các đỉnh trong đoạn [L;R] là vùng “*trung lập*”.

Ngược lại, khi H(S) ∩ W(E) = ∅ thì không có đỉnh trung lập nào kết nối giữa hai vùng liên thông H(S) và W(E) nên không thể có đườn đi S → ? → E.

Thuật toán được phác thảo như sau:

Bước 1. Xác định hai đồ thị con H và W. Độ phức tạp O(R) , O(N-L)

Bước 2. Xác định các miền liên thông chứa điểm xuất phát S của H, ký hiệu là H(S). O((N-L)N)

Bước 3. Xác định các miền liên thông chứa điểm kết thúc E của W, ký hiệu là W(E). O(RN)

Bước 4. Kết luận:

* Xác định giao T của H(S) và W(E).
* Nếu T là rỗng: trả lời vô nghiệm; ngược lại: trả lời có đường đi từ S tới E.

#### Input

Dữ liệu vào được ghi trong file text WEREWOLF.INP với các dòng như sau:

2 số N và M

M giá trị x[0],…,x[M−1]

M giá trị y[0],…,y[M−1]

Q test

Mỗi test được cho dưới dạng 4 số

S, E, L, R

Các dãy số x[j] và y[j] cho biết đường thứ j nối thành phố x[j] với thành phố y[j], 0 ≤ j < M.

Các số cùng dòng được ghi cách nhau.

#### Giới hạn

A number and numbers on a white background

Description automatically generated

#### Chú ý

Trong bài này các chỉ số đỉnh được gán trị từ 0 đến N−1, các chỉ số cạnh được gán trị từ 0 đến M−1 và các test được gán thứ tự từ 0 đến Q−1.

#### Ví dụ

|  |  |  |
| --- | --- | --- |
| WEREWOLF.INP |  | A diagram of a network  Description automatically generated |
| 6 6  5 1 1 3 3 5  1 2 3 4 0 2  3  4 2 1 2  4 2 2 2  5 4 3 4 | *Đồ thị gồm 6 đỉnh, 6 cạnh.*  *Các đỉnh được mã số từ 0 đến 5.* |
| *6 cạnh: (5,1), (1,2), (1,3), (3,4), (3,0), (5,2)* |
| *3 test* |
| *Test 0: S = 4, E = 2, L = 1, R = 2.* |
| *Test 1: S = 4, E = 2, L = 2, R = 2.* |
| *Test 2: S = 5, E = 4, L = 3, R = 4.* |

#### Giải trình Test 0

|  |
| --- |
| Test 0: S = 4, E = 2, L = 1, R = 2 |
| A grid of numbers and letters  Description automatically generated |

|  |  |
| --- | --- |
| Test 0: S = 4, E = 2, L = 1, R = 2 | |
| A black circle with black lines and a number  Description automatically generated with medium confidence | A diagram of a network  Description automatically generated |
| *Các vùng di chuyển của Sói* | *Các vùng di chuyển của Người* |

Test 0 chia đồ thị G thành hai đồ thị con H và W.

* H: các vùng dành cho Người:

Đỉnh: (1), (2), (3), (4), (5).

Cạnh: (1,2), (1,3), (1,5), (2,5), (3,4).

H có 1 miền liên thông: {1,2,3,4,5}.

Miền liên thông chứa điểm xuất phát S = 4: H(4) = {1,2,3,4,5}

* W: các vùng dành cho Sói.

Đỉnh: (0), (1), (2).

Cạnh: (1,2).

W có 2 miền liên thông: {0}, {1,2}.

Miền liên thông chứa đỉnh kết thúc E = 2: W(2) = {1,2}

T = H(4) ∩ W(2) = {1, 2}

Với Test này bạn di chuyển như sau:

Xuất phát tại thành phố (H4): bạn ở hình dáng người (H4).

Từ (H4) bạn di chuyển đến (H3): bạn ở hình dáng người (H4) → (H3).

Từ (H3) bạn di chuyển đến (H1): bạn ở hình dáng người (H3) → (H1).

Tại (H1) bạn chuyển sang lốt sói : (H1)/(W1).

Từ (W1) bạn di chuyển đến (W2) trong hình dáng sói: (W1) → (W2).

Sơ đồ di chuyển được viết như sau:

(H4) → (H3) → (H1)/(W1) → (W2)

Bạn cũng có thể di chuyển đến (H2) rồi mới chuyển đổi lốt:

(H4) → (H3) → (H1) → (H2)/(W2)

#### Chú ý

Chỉ số trong bài này biến thiên trong khoảng 0..N

#### Giải trình Test 1

A white grid with black letters and numbers

Description automatically generated

|  |  |
| --- | --- |
| Test 1: S = 4, E = 2, L = 2, R = 2 | |
| A diagram of a diagram  Description automatically generated with medium confidence | A diagram of a network  Description automatically generated |
| *Các vùng di chuyển của Sói* | *Các vùng di chuyển của Người* |

Test 1 chia đồ thị G thành hai đồ thị con H và W.

* H: các vùng dành cho Người:

Đỉnh: (2), (3), (4), (5).

Cạnh: (2,5), (3,4).

H có 2 miền liên thông: {2,5}, {3,4}.

Miền liên thông chứa điểm xuất phát S = 4: H(4) = {3,4}

* W: các vùng dành cho Sói.

Đỉnh: (0), (1), (2).

Cạnh: (1,2).

W có 2 miền liên thông: {0}, {1,2}.

Miền liên thông chứa đỉnh kết thúc E = 2: W(2) = {1,2}

T = H(4) ∩ W(2) = ∅

Bạn xuất phát từ đỉnh H(4) và chỉ có thể di chuyển đến H(3) chứ không thể di chuyển đến W(2). Vậy Test 1 vô nghiệm.

#### Giải trình Test 2

A white grid with black numbers and letters

Description automatically generated

A diagram of a number

Description automatically generated with medium confidence

Test 2 chia đồ thị G thành hai đồ thị con H và W.

* H: các vùng dành cho Người:
* Đỉnh: (3), (4), (5).
* Cạnh: (3,4).
* H có 2 miền liên thông: {5}, {3,4}.
* Miền liên thông chứa điểm xuất phát S = 5: H(5) = {5}.
* W: các vùng dành cho Sói.

Đỉnh: (0), (1), (2), (3), (4).

Cạnh: (0,3), (1,2), (1,3), (3,4).

W có 1 miền liên thông: {0,1,2,3,4}.

Miền liên thông chứa đỉnh kết thúc E = 4: W(4) = {0,1,2,3,4}

T = H(5) ∩ W(4) = ∅

Bạn xuất phát từ đỉnh cô lập H(5) nên không thể di chuyển đến bất kỳ đỉnh nào khác. Vậy Test 2 vô nghiệm.

#### Cài đặt

Bài này dùng kỹ thuật Find-Union. Chi tiết kỹ thuật này bạn đọc có thể xem trong các bài đã trình bày ở phần trên, đặc biệt là bài Subgraph.

Ý tưởng của thuật toán khá đơn giản. Hai giới hạn trái L và phải R chia đồ thị G ban đầu thành hai đồ thị con: 0..L..R..N-1

Đồ thị con H dành cho sự di chuyển của Người. H bao gồm các đỉnh L..N−1 và các cạnh (x,y) có hai đỉnh nằm trọn trong H: L ≤ x,y ≤ N−1.

Đồ thị con W dành cho sự di chuyển của Sói. W bao gồm các đỉnh 0..R và các cạnh (x,y) có hai đỉnh nằm trọn trong W: 0 ≤ x,y ≤ R.

Trong đồ thị con H bạn dùng kỹ thuật Find-Union tách ra miền liên thông H(S) chứa đỉnh xuất phát (S). Theo bài Subgraph, ta dùng hàm ConnectedZone(L, N-1, S) với độ phức tạp

Tương tự, trong đồ thị con W bạn dùng kỹ thuật Find-Union tách ra miền liên thông W(E) chứa đỉnh kết thúc (E). Theo bài Subgraph, ta dùng hàm ConnectedZone(0, R, E) với độ phức tạp

Nếu H và W giao nhau thì có đường tứ S đến E; ngược lại là không có đường. Tìm giao của hai tập đonhr có đọ phức tạp O(N).

#### Program

# WEREWOLF

def Go():

if input(' ? ') == '.': exit(0)

def Find(x):

while d[x] != x: x = d[x]

return x

def Union(x, y):

global d

x, y = Find(x), Find(y)

if x == y: return 0

if x < y: d[y] = x

else: d[x] = y

return 1

def Less(s, e): return set(filter(lambda x: x <= e, s))

# TPLT chứa v của subgraph(s..e)

def ConnectedZone(s, e, v):

global d

d = list(range(n))

r = range(s,e+1)

for x in r:

for y in Less(vt[x], e): Union(x,y)

for i in r: d[i] = Find(i)

return set(j for j in r if d[j] == d[v])

def Run():

global n, vt

with open('werewolf.inp') as f:

n, m = map(int, f.readline().strip().split())

print(n,'đỉnh', m, 'cạnh')

sx = list(map(int, f.readline().strip().split()))

sy = list(map(int, f.readline().strip().split()))

vt = {i: set({}) for i in range(n)}

for i in range(m): # đọc các cạnh

x, y = sx[i], sy[i]

if x == y: continue

if x < y: vt[x].add(y)

else: vt[y].add(x)

q = int(f.readline())

print(q, 'test')

for t in range(0, q):

print(' Test', t)

S, E, L, R = map(int, f.readline().strip().split())

print(' SELR:', S, E, L, R)

# Tim vung lien thong chua dinh S

# trong subgraph H: L..(n-1)

# 0.....L....R.....(n-1)

HS = ConnectedZone(L,n-1, S)

print(' TPLT chứa', S, ': ', HS)

# Tim vung lien thong chua dinh E

# trong subgraph W: 0..R

# 0.....L....R.....(n-1)

WE = ConnectedZone(0, R, E)

print(' TPLT chứa', E, ': ', WE)

# T = HS giao WE

if len(HS & WE) > 0: print(' YES.')

else: print(' NO.')

# application

Run()

#### Output

6 đỉnh 6 cạnh

So test 3

Test 0

SELR: 4 2 1 2

TPLT chứa 4 : {1, 2, 3, 4, 5}

TPLT chứa 2 : {1, 2}

YES.

Test 1

SELR: 4 2 2 2

TPLT chứa 4 : {3, 4}

TPLT chứa 2 : {1, 2}

NO.

Test 2

SELR: 5 4 3 4

TPLT chứa 5 : {5}

TPLT chứa 4 : {0, 1, 2, 3, 4}

NO.