Dudu Service Maker

Big-O Blue Lecture 6: DFS

Tóm tắt đề bài

Tóm tắt đề bài

- Dudu nhận thấy rằng để hiểu được tài liệu này thì phải biết được kiến thức từ tài liệu khác.
- Tuy nhiên thì trong những tài liệu như vậy cũng có thể tạo ra một vòng tròn. Ví dụ tài liệu A cần hiểu tài liệu B trước nhưng tài liệu B cũng cần phải hiểu tài liệu A trước.
- Yêu cầu: cho danh sách liên kết giữa các tài liệu, hãy kiểm tra xem trong các tài liệu này có một chuỗi vòng tròn hay không.

Mô tả Input/Output

Input: Dòng đầu tiên chứa một số nguyên T $(1 \leq T \leq 100)$ cho biết số test. Trong T test case tiếp theo:

- Dòng đầu tiên của test chứa hai số nguyên N và M $(1 \leq N \leq 10^4; 1 \leq M \leq 3 \times 10^4)$. Lần lượt là số tài liệu và số mối liên hệ giữa hai tài liệu.
- M dòng kế tiếp cho các số A và B $(1 \leq A, B \leq N; A \neq B)$ cho biết tài liệu A cần phải hiểu tài liệu B trước.

  Lưu ý: có thể xuất hiện một cặp A và B nhiều lần.

Ouput: Với mỗi test case thì in ra "YES" nếu có một chuỗi vòng trong test. Ngược lại thì in ra là "NO".

Giải thích ví dụ

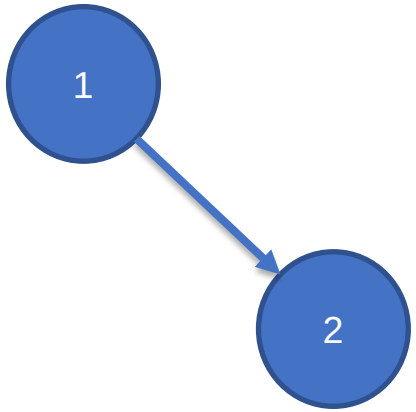
Ví dụ 1

Test case 1:

N = 2 và M = 1

1 2

Output : NO



Ví dụ 2

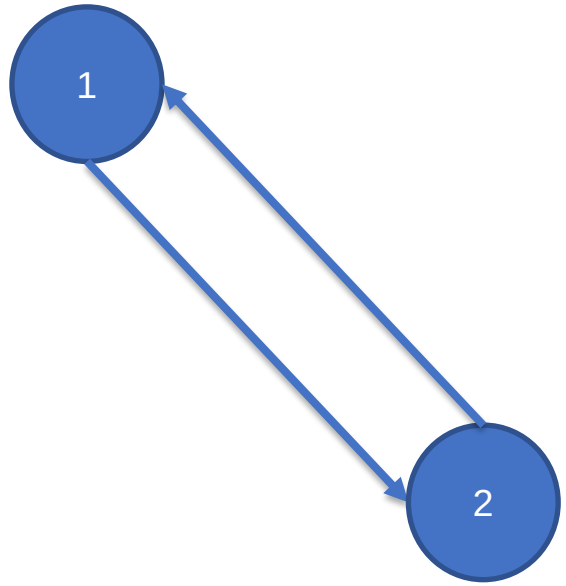
Test case 2:

N = 2 và M = 2

1 2

2 1

Output : YES



Ví dụ 3

Test case 3:

N = 4 và M = 4

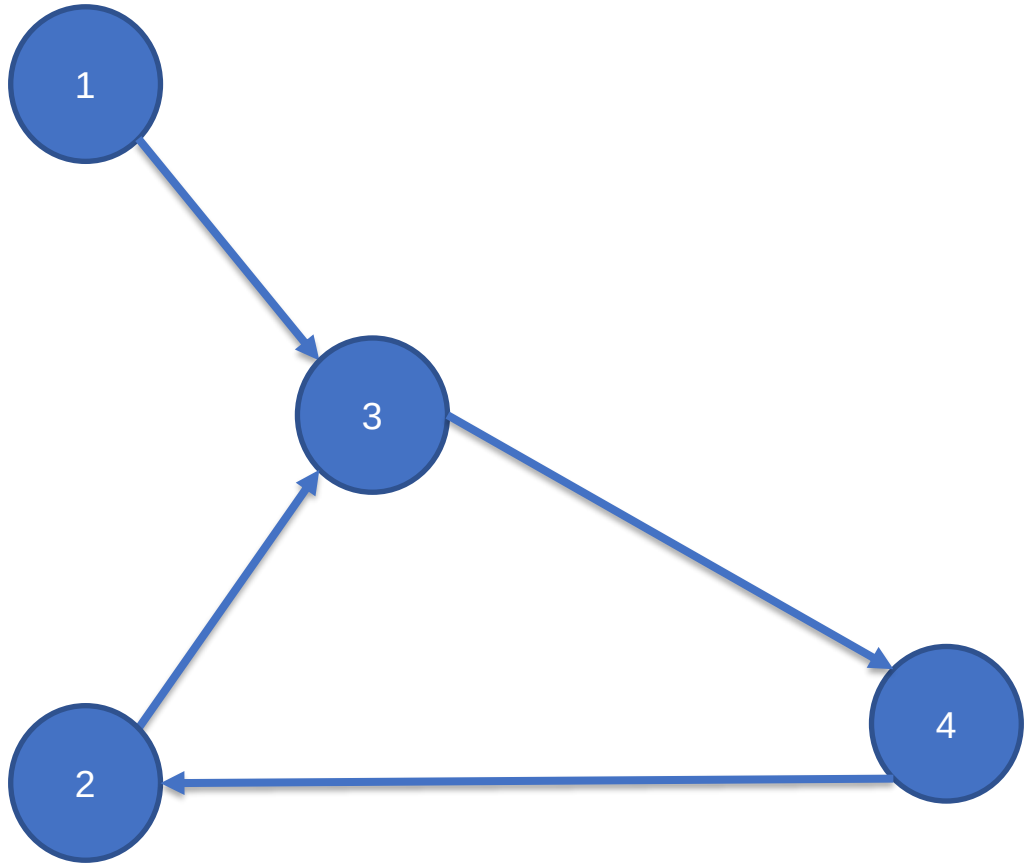
2 3

3 4

4 2

1 3

Output : YES

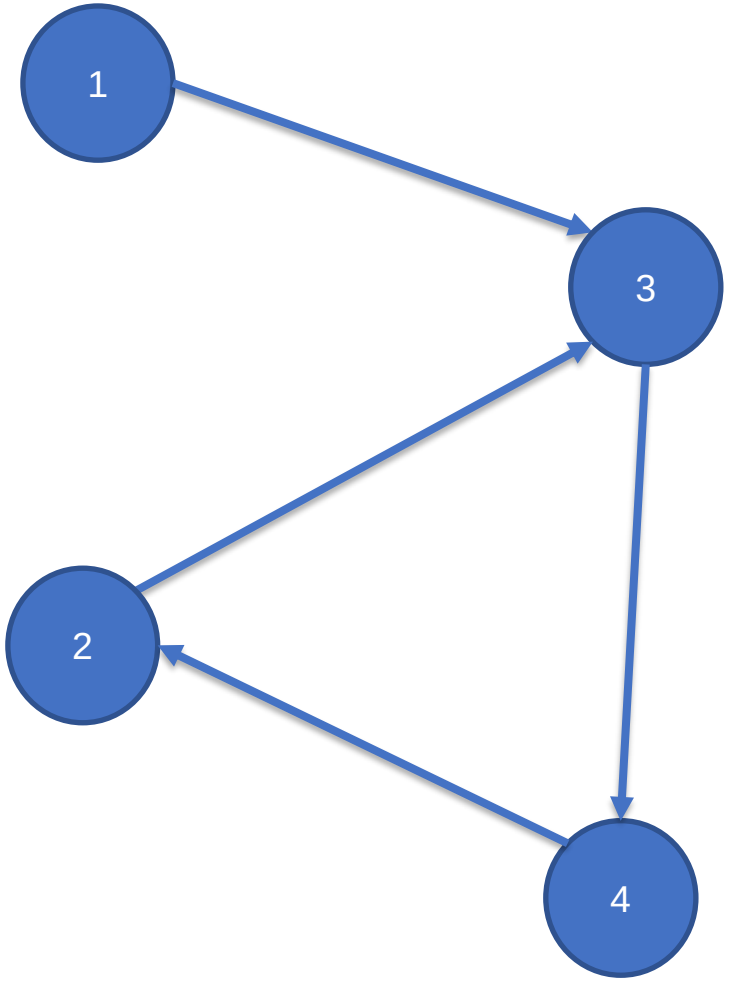


Hướng dẫn giải

Nhận xét

- Nếu xem mỗi tài liệu là một đỉnh và việc "tài liệu A cần tài liệu B" là một cạnh có hướng từ A → B thì bài toán sẽ thành kiểm tra xem trong đồ thị có hướng của đề cho có chu trình hay không (như hình bên).

→ Sử dụng DFS để tìm kiếm chu trình trong đồ thị.



Tìm kiếm chu trình bằng DFS?

- Mình có thể bắt đầu tại tất cả các đỉnh và DFS. Đánh dấu các đỉnh ta đang đi qua bằng mảng.
- Xét với mỗi đỉnh **u**, đánh dấu đỉnh và kiểm tra:
  - Nếu có một **đỉnh kề v** với đỉnh **u** mà đồng thời cũng trong đường đi ta đã đi qua thì suy ra đã tìm ra chu trình trong đồ thị.
  - Nếu mà có **đỉnh kề v** với đỉnh **u** mà không có trong đường đi thì mình sẽ duyệt tiếp DFS đỉnh v đó.
- Sau khi duyệt với mọi **đỉnh kề v** rồi thì mình sẽ bỏ đánh dấu lại đỉnh **u**.

Mô phỏng thuật toán

Test mẫu:

N = 6 và M = 6

2 5

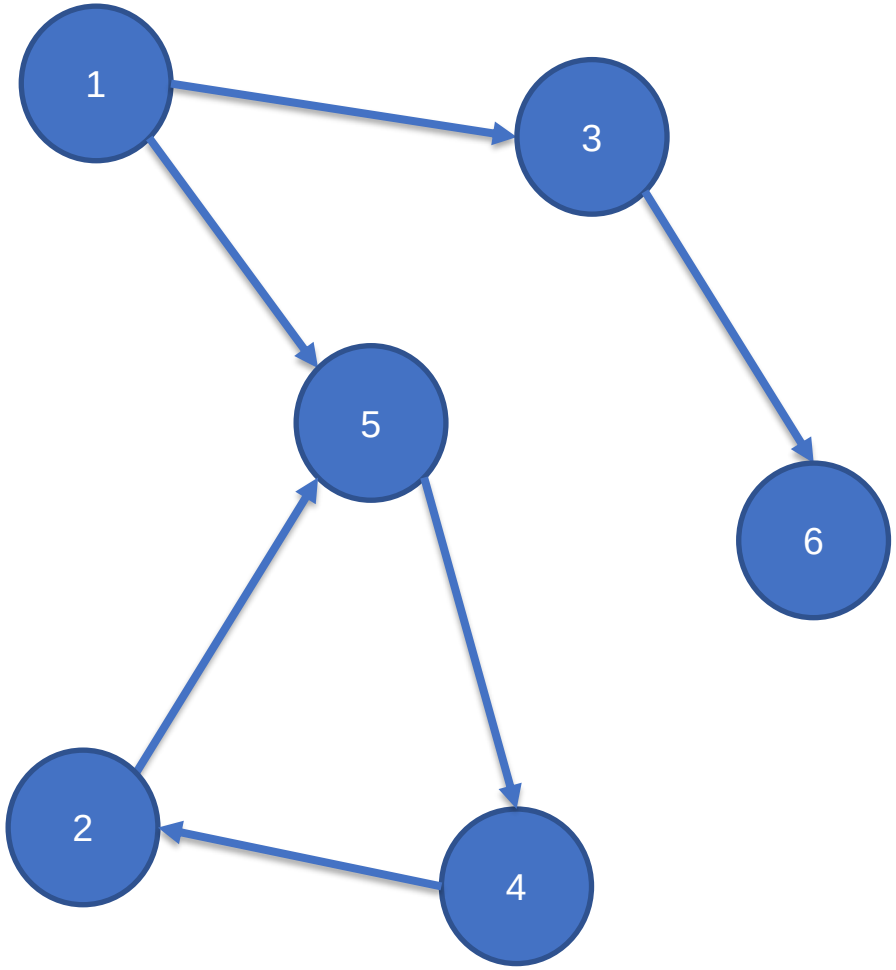
5 4

4 2

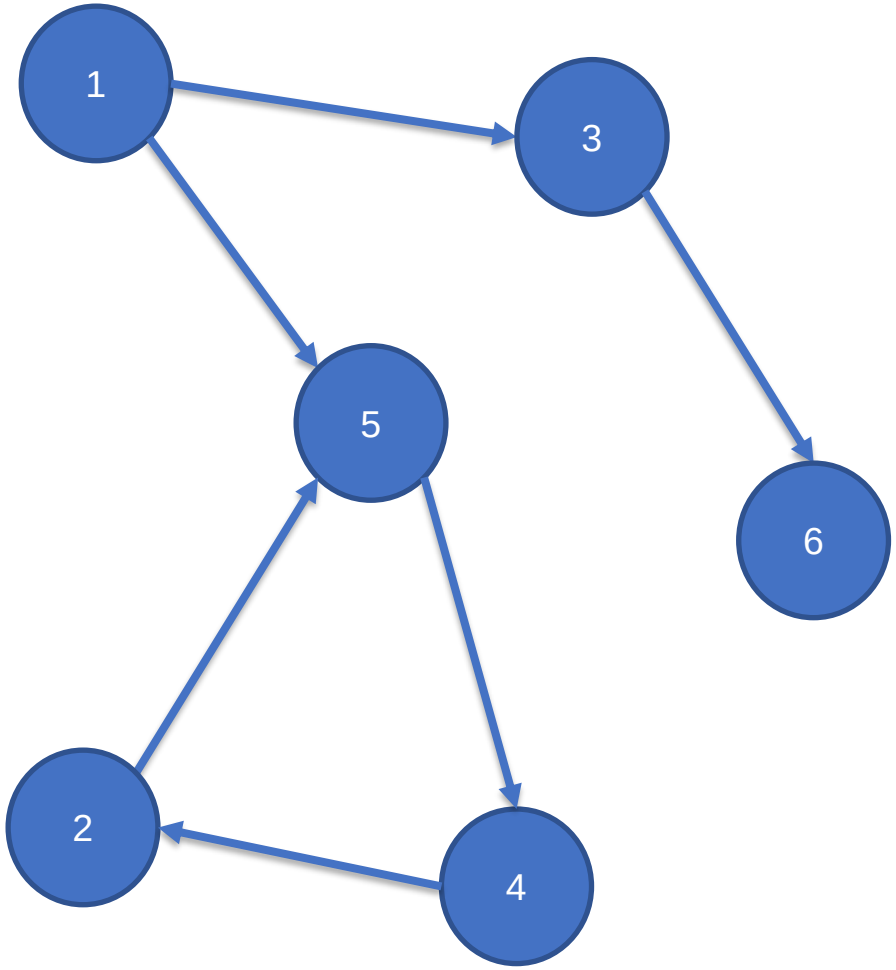
1 5

1 3

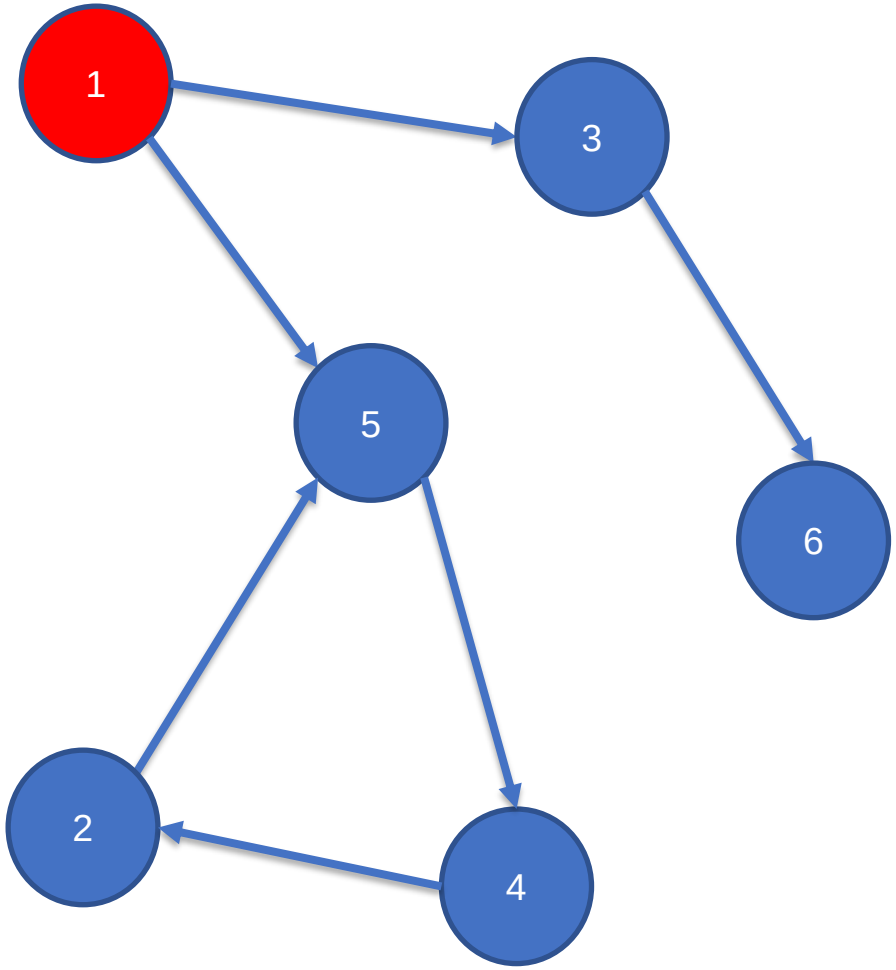
3 6



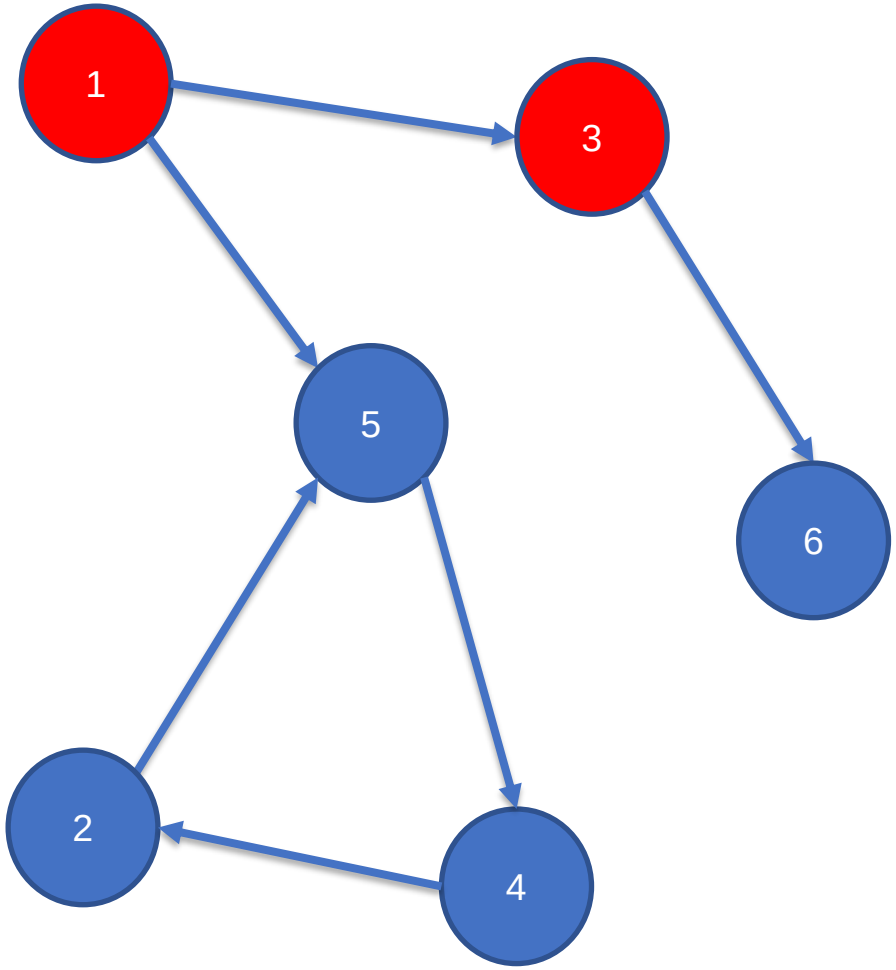
Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	False	False	False	False	False	False



Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	False	False	False	False	False



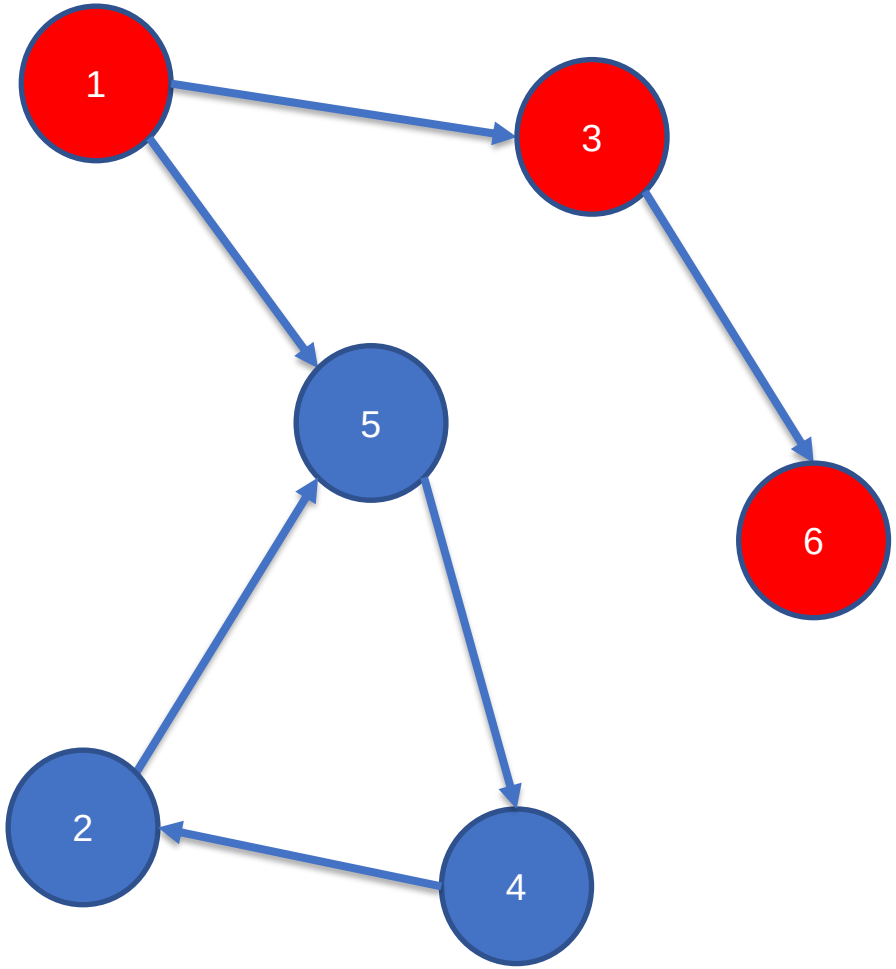
Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	False	True	False	False	False



Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	False	True	False	False	True

Đến đây, do đỉnh 6 không có đỉnh kề nó.

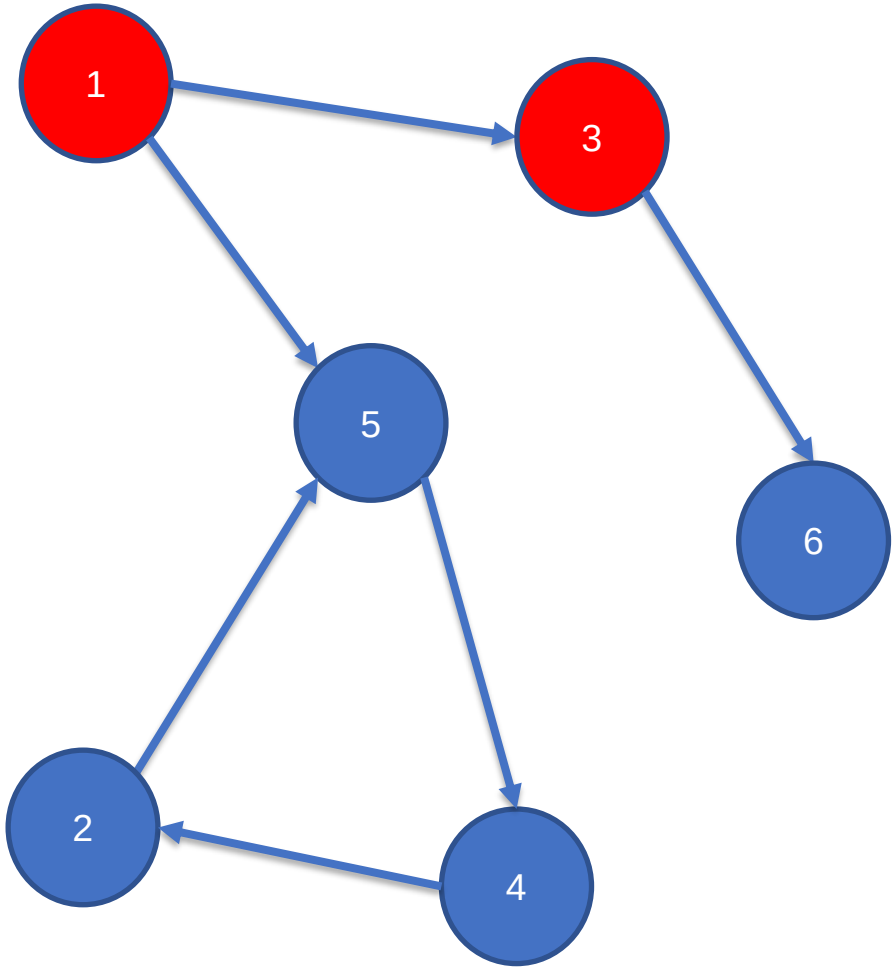
➔Chúng ta quay lui về đỉnh số 3.



Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	False	True	False	False	False

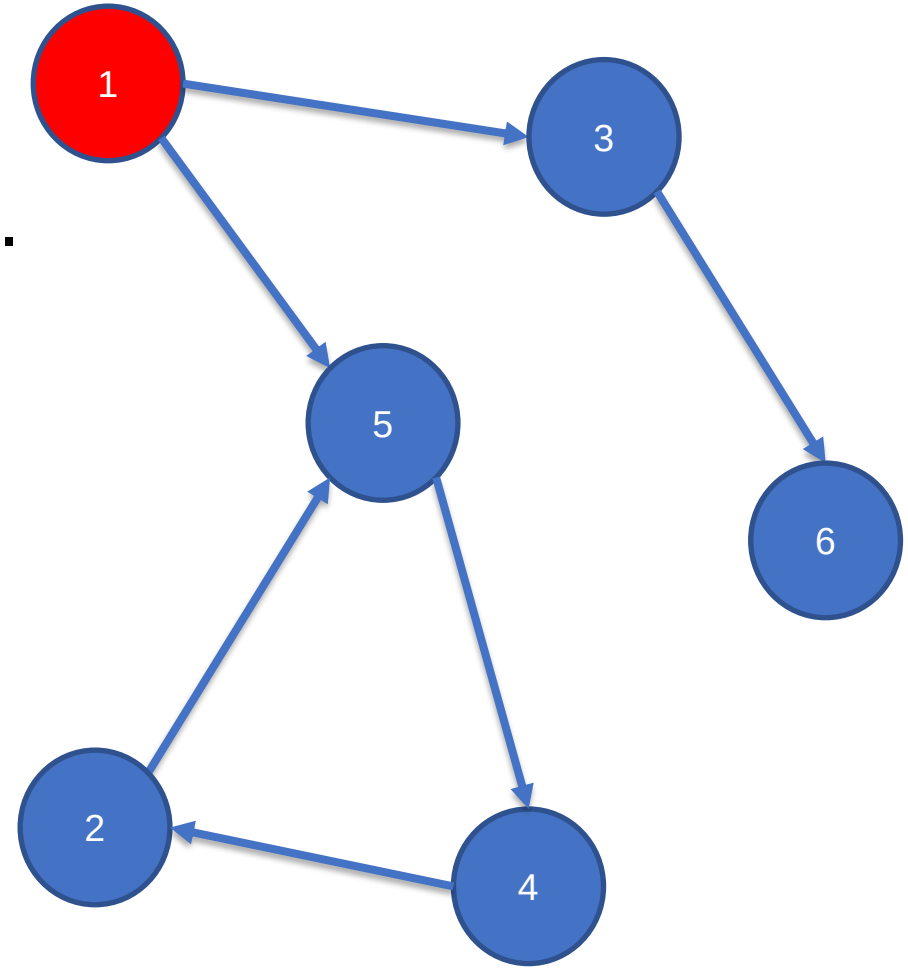
Tương tự, đỉnh 3 không còn đỉnh kề khác.

➔Chúng ta quay lui về đỉnh số 1.

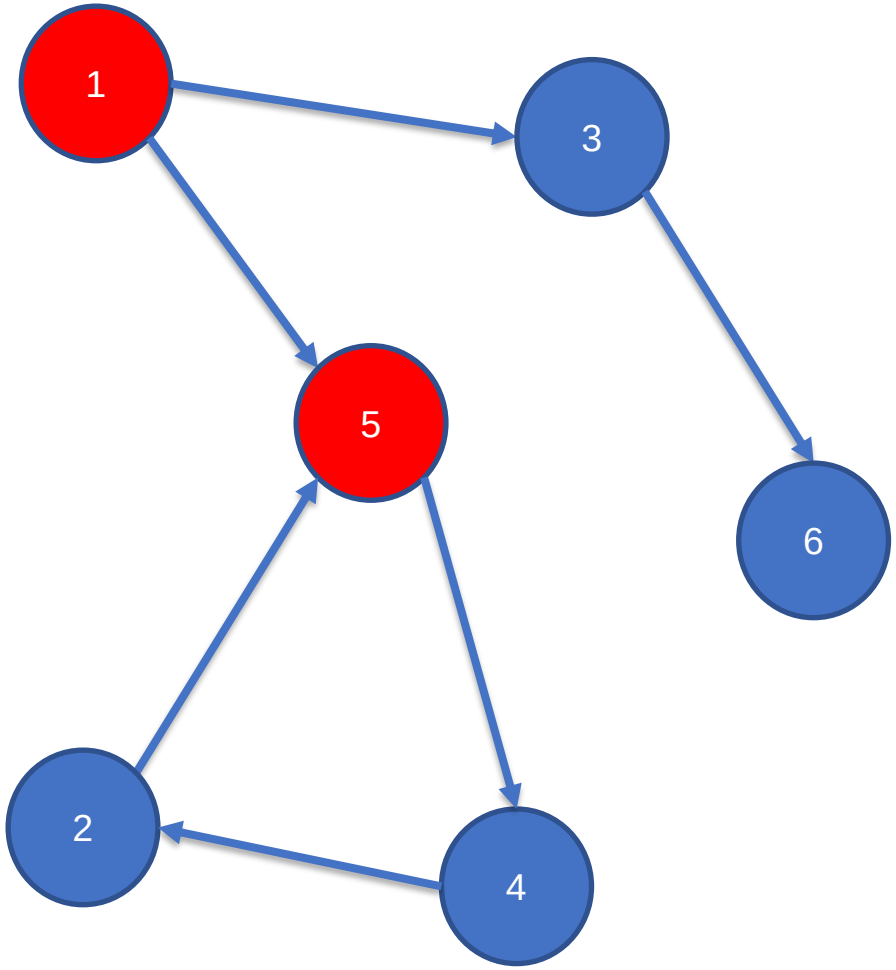


Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	False	False	False	False	False

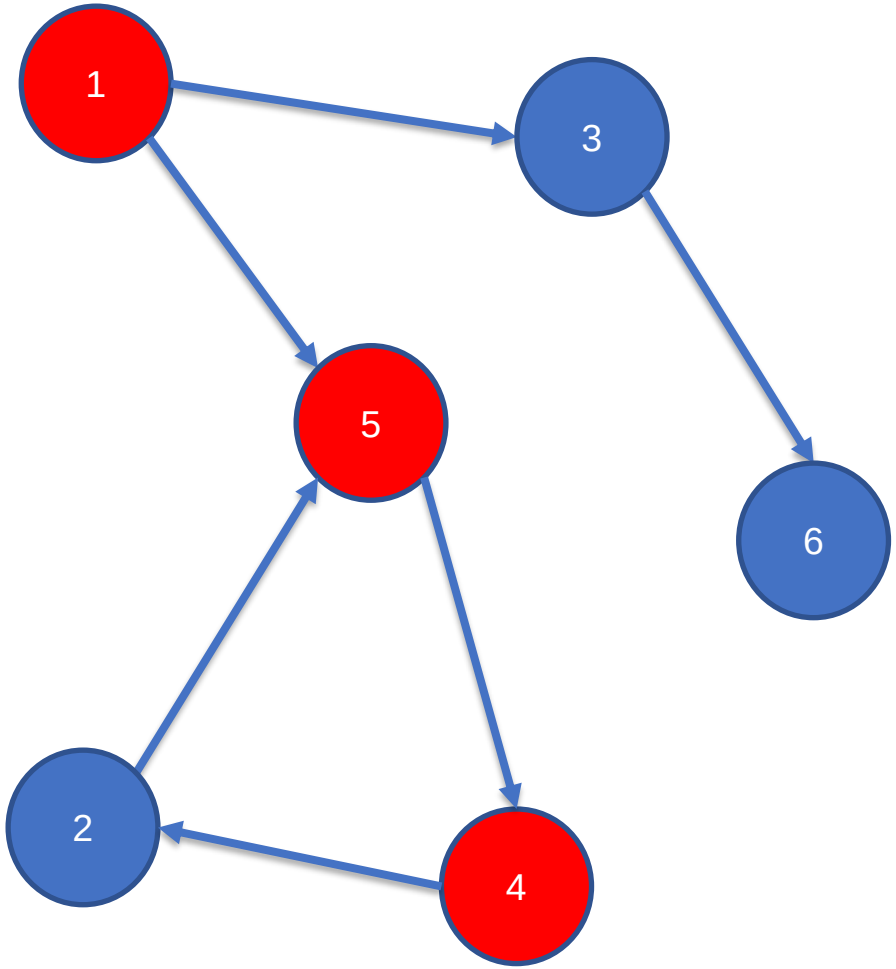
Đỉnh 1 sẽ duyệt sang tiếp đỉnh kề 5.



Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	False	False	False	True	False



Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	False	False	True	True	False

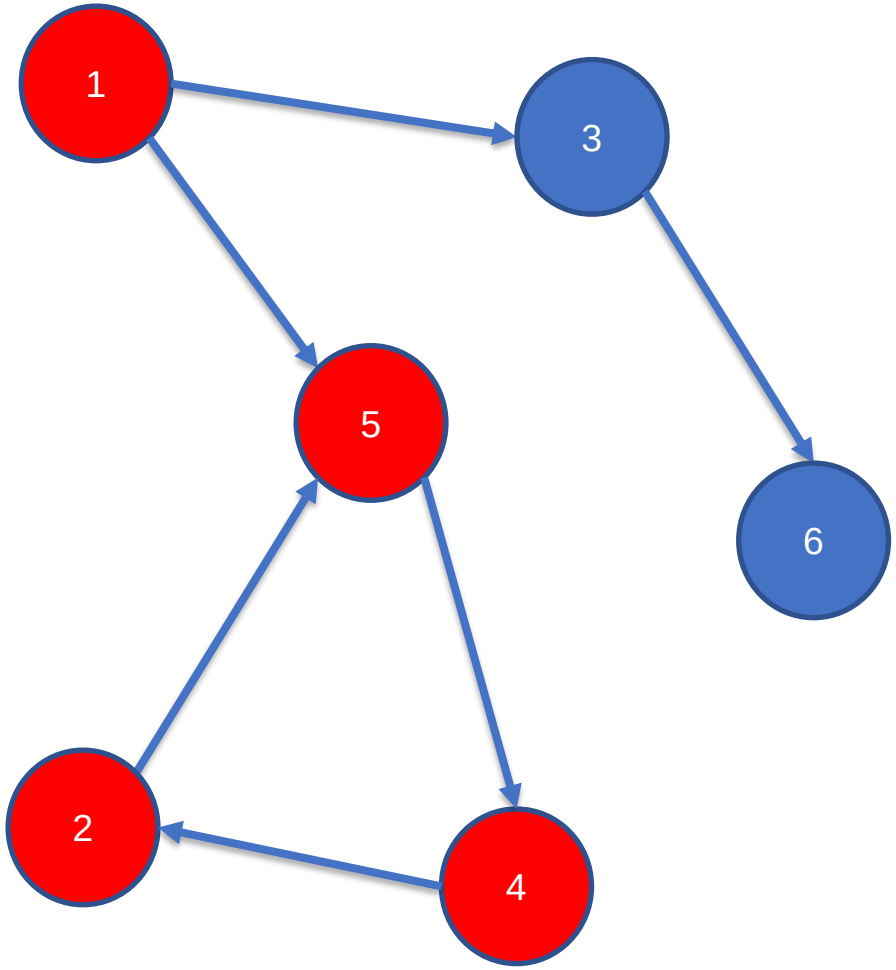


Index	1	2	3	4	5	6
Graph	[3, 5]	[5]	[6]	[2]	[4]	[]
Flag	True	True	False	True	True	False

Đến đây, do đỉnh 2 có đỉnh kề nó là đỉnh 5 và 5 cũng đang được đánh dấu.

➔ Chúng ta đã tìm ra được chu trình trong đồ thị này.

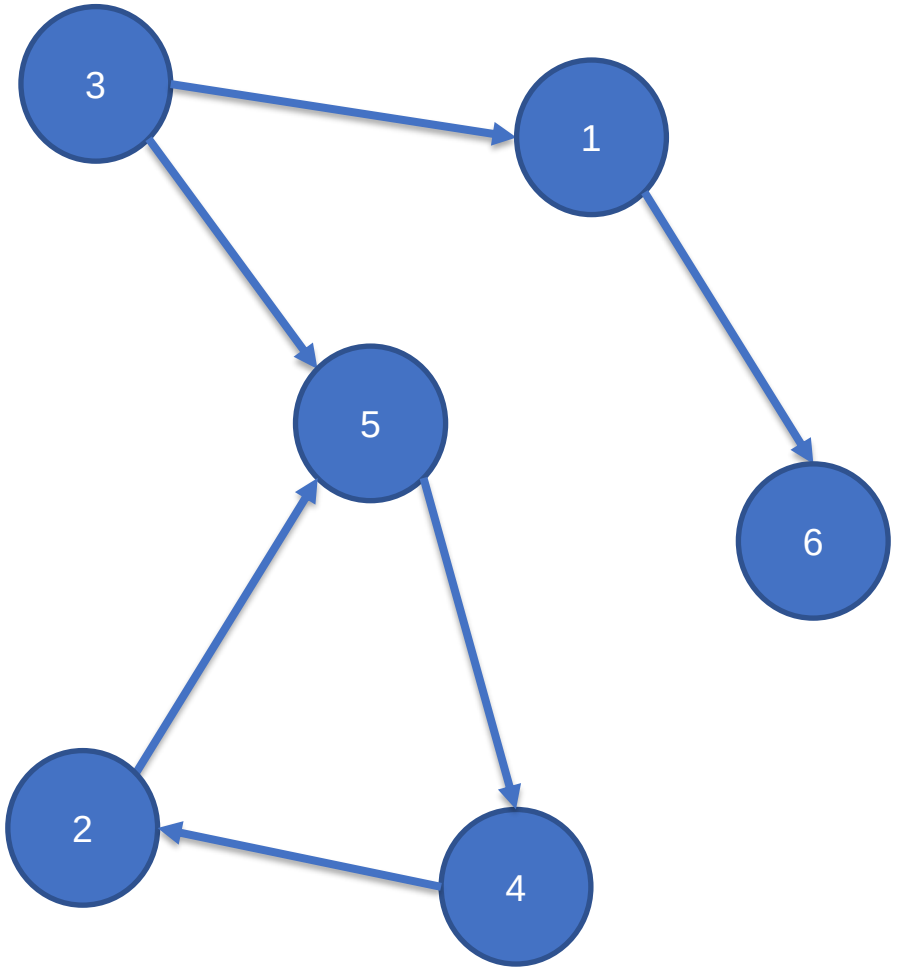
➔ Thuật toán dừng lại và xuất ra đáp án



Vài nhận xét thêm

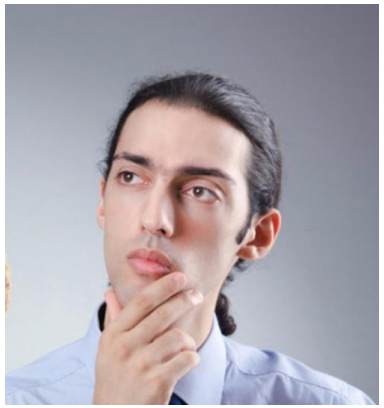
- Có thể dễ thấy, nếu mình chỉ bắt đầu duyệt DFS từ một đỉnh duy nhất (ví dụ đỉnh 1) thì có thể sẽ không kiếm ra đáp án (ví dụ như hình bên).
- Do đó, phải bắt đầu duyệt từ mọi đỉnh để có thể kiếm ra chu trình.

➔ Tuy nhiên, như vậy độ phức tạp thời gian của tụi mình sẽ lên đến $O(N \times (N + M))$ và bị TLE.



Vài nhận xét thêm

- Vậy có cách nào để mà tối ưu thời gian?

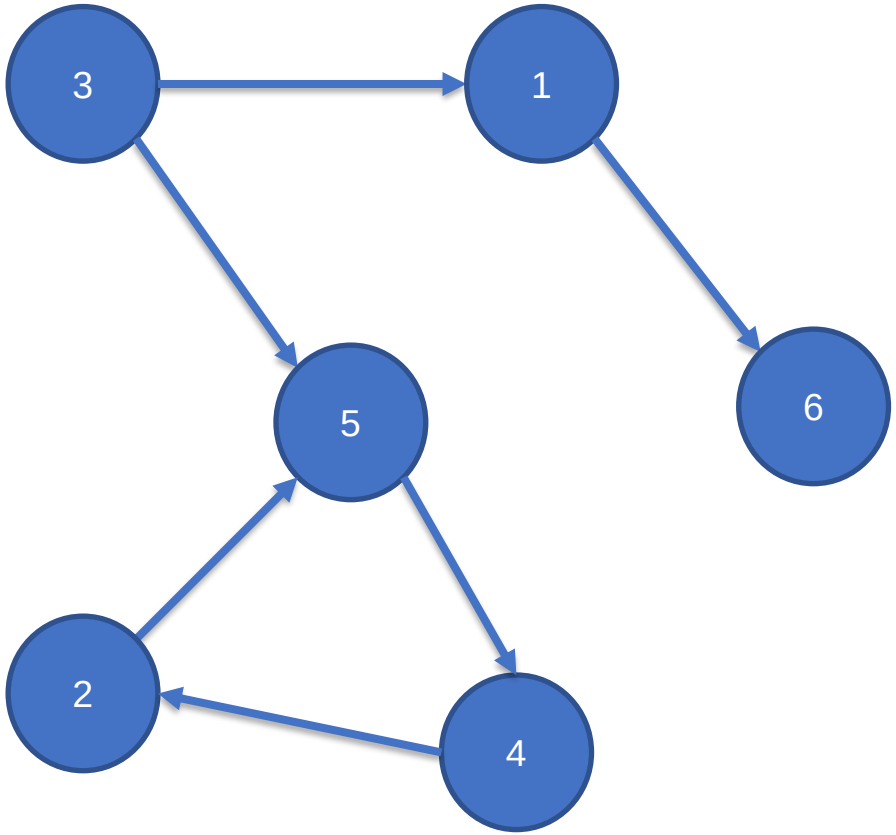


➔Nếu mình biết trước là nếu duyệt DFS một đỉnh u mà không có chu trình thì mình có thể bỏ qua đỉnh u đó.

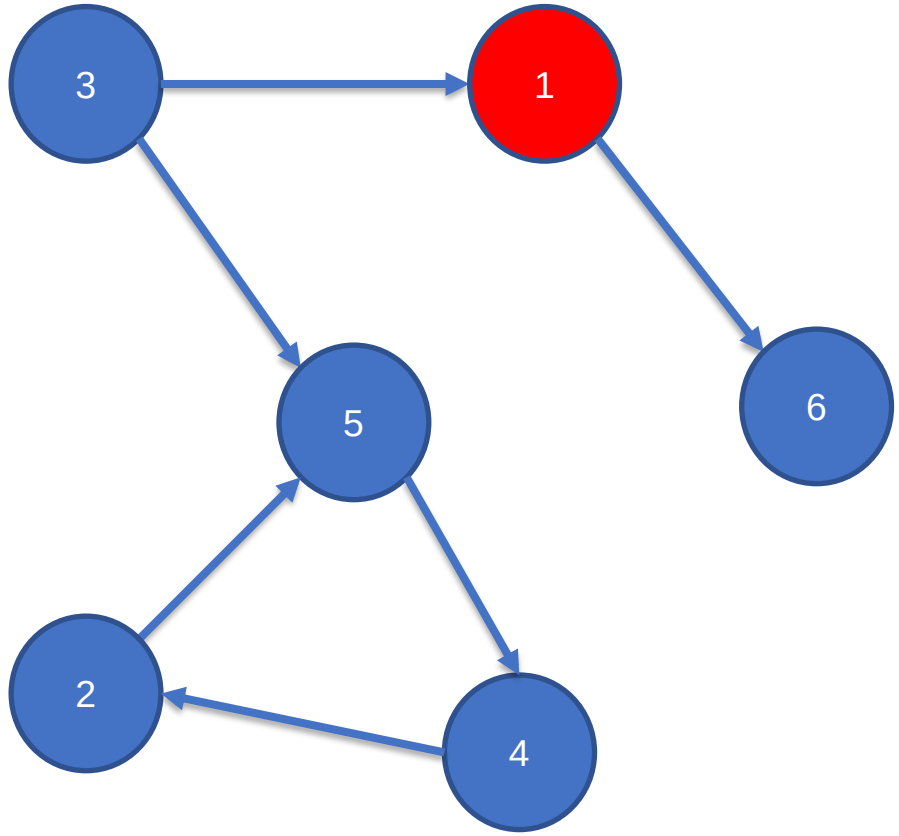
➔Sử dụng thêm một mảng lưu trữ Visited[i] = True nếu mà đỉnh i đã được duyệt DFS.

➔Như thế mỗi đỉnh được duyệt qua đúng một lần → Độ phức tạp thời gian $O(N+M)$

Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	False	False	False	False	False	False
Visited	False	False	False	False	False	False



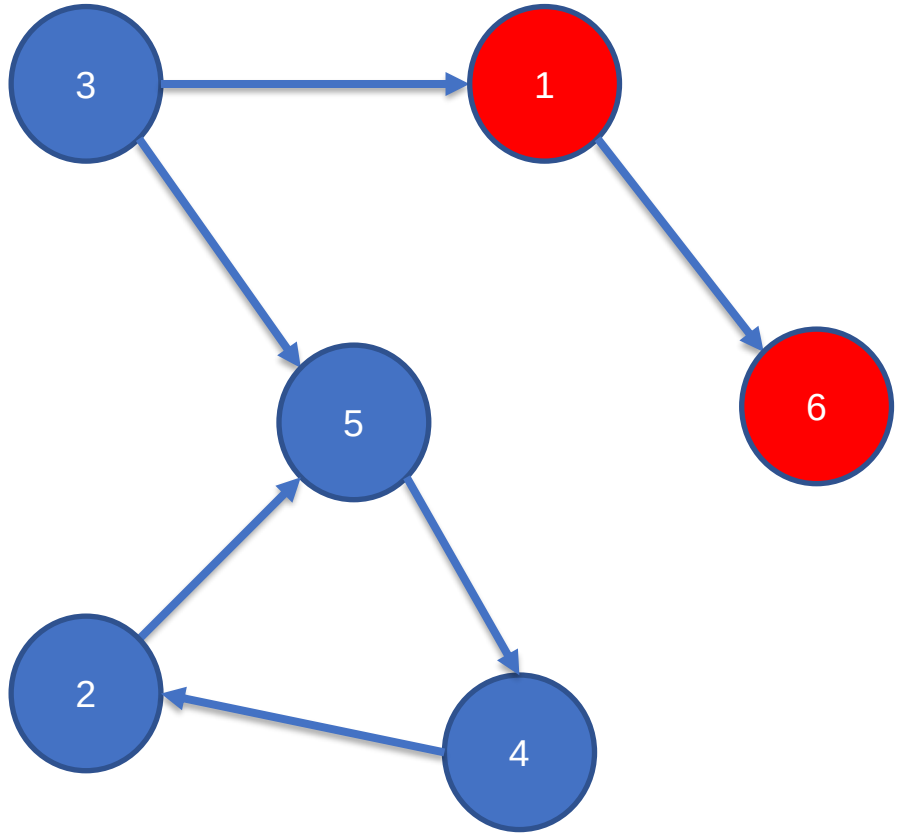
Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	True	False	False	False	False	False
Visited	True	False	False	False	False	False



Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	True	False	False	False	False	True
Visited	True	False	False	False	False	True

Đến đây, do đỉnh 6 không có đỉnh kề nó.

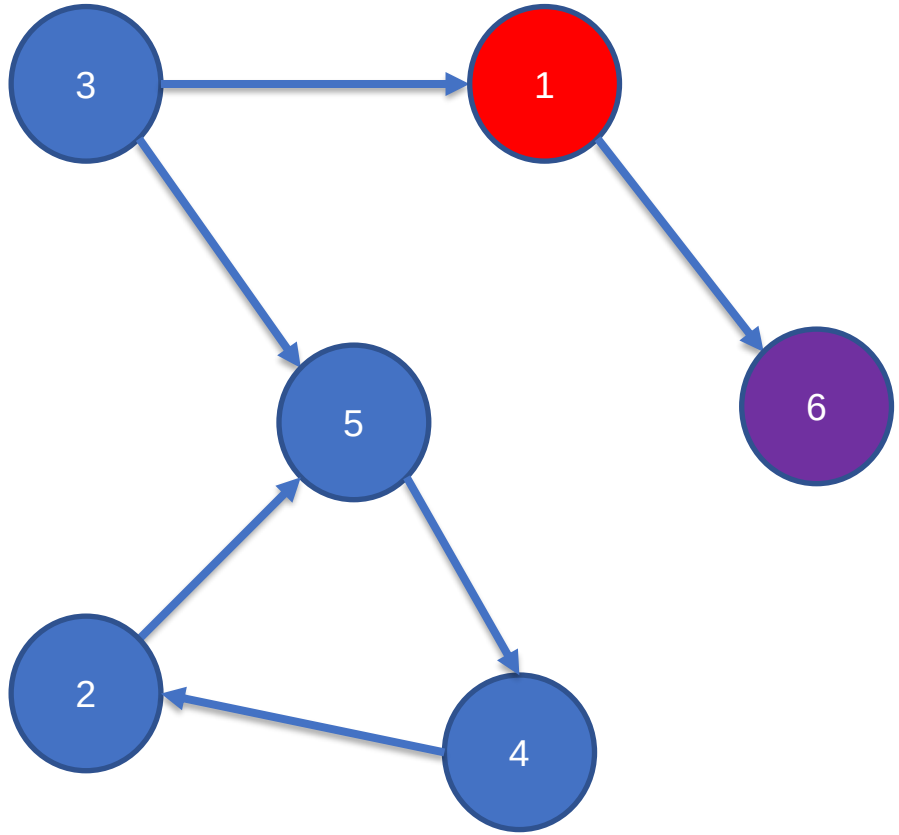
➔Chúng ta quay lui về đỉnh số 1.



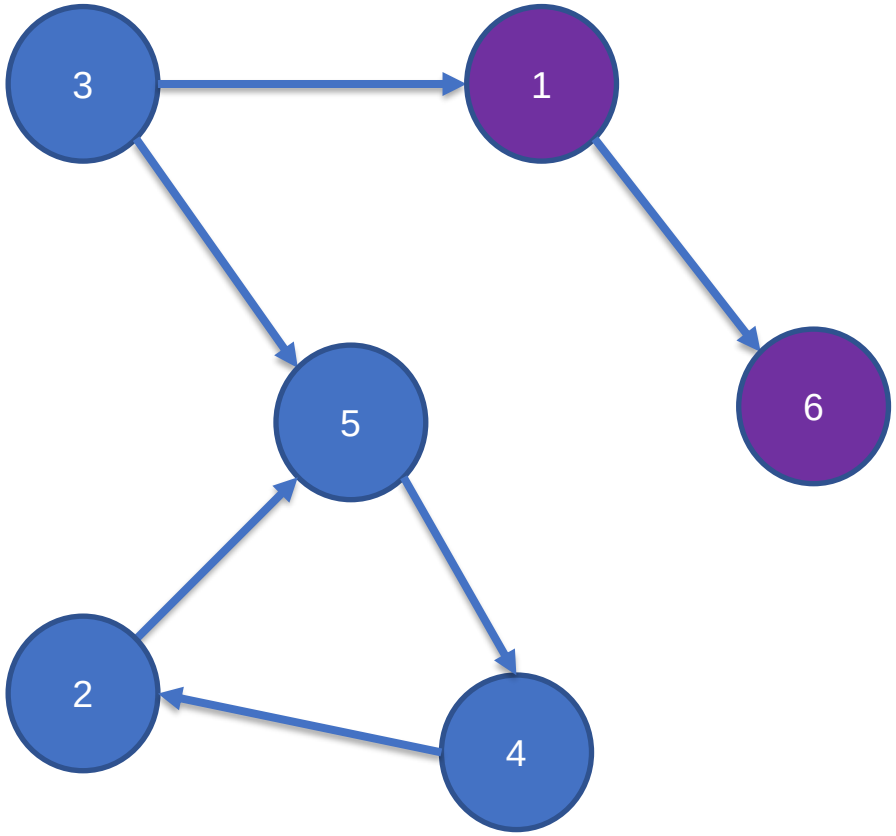
Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	True	False	False	False	False	False
Visited	True	False	False	False	False	True

Đến đây đỉnh 1 không còn đỉnh kề khác

➔Kết thúc DFS tại đỉnh 1.



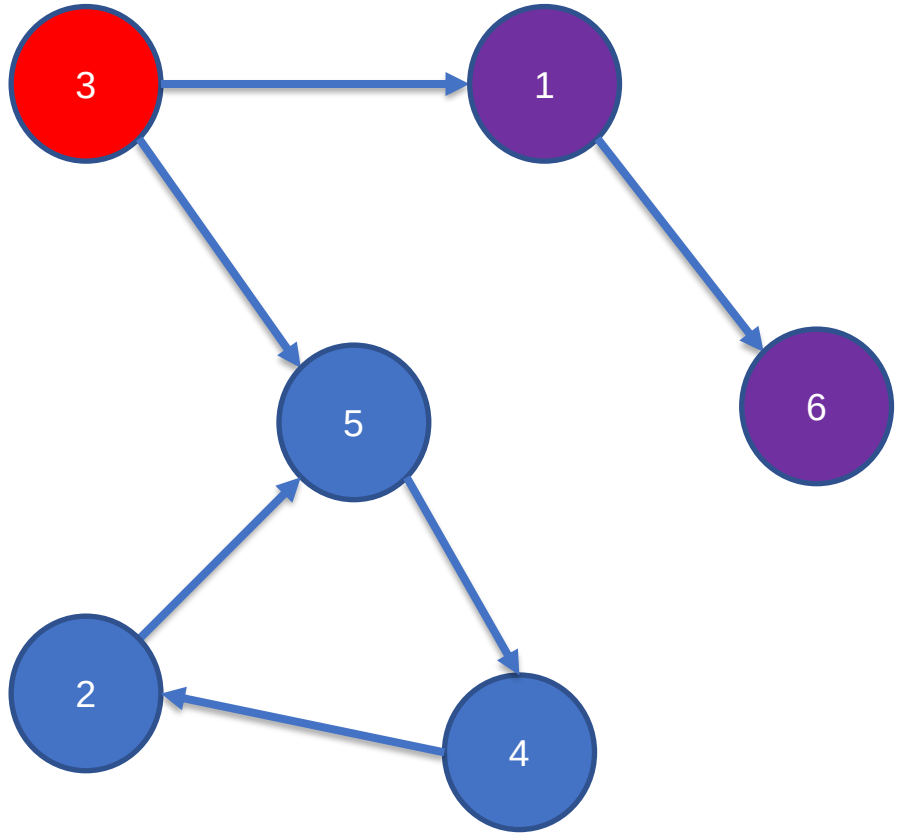
Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	False	False	False	False	False	False
Visited	True	False	False	False	False	True



Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	False	False	True	False	False	False
Visited	True	False	True	False	False	True

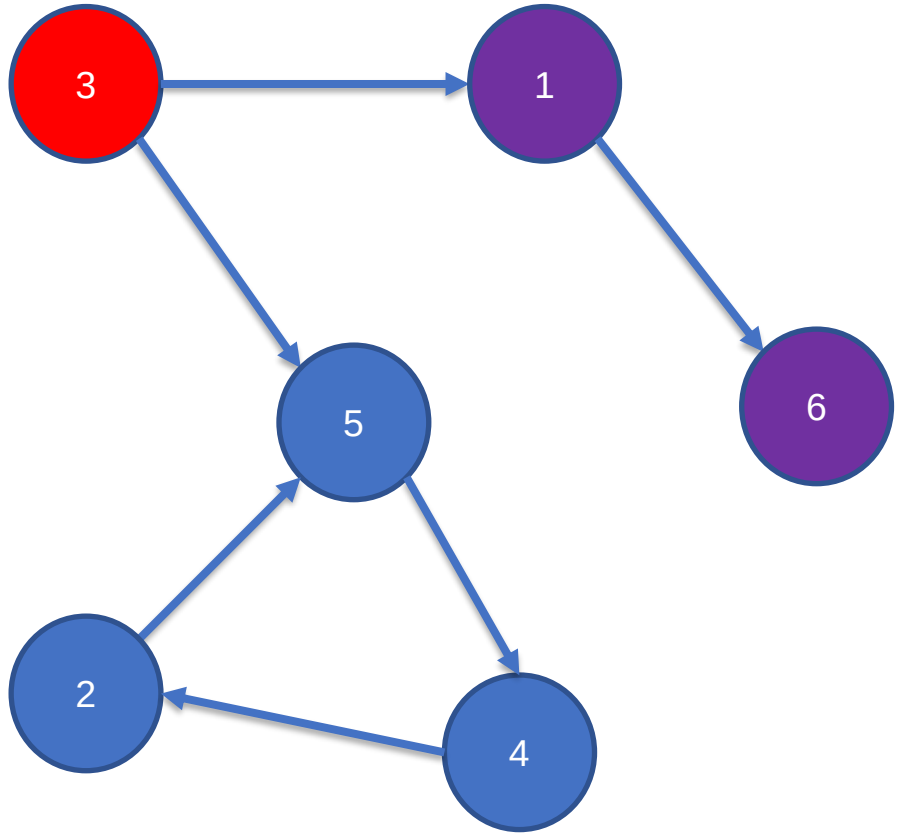
Duyệt đỉnh kề thấy đỉnh số 1 đã được Visited[1] = True

➔ Không DFS đỉnh 1.

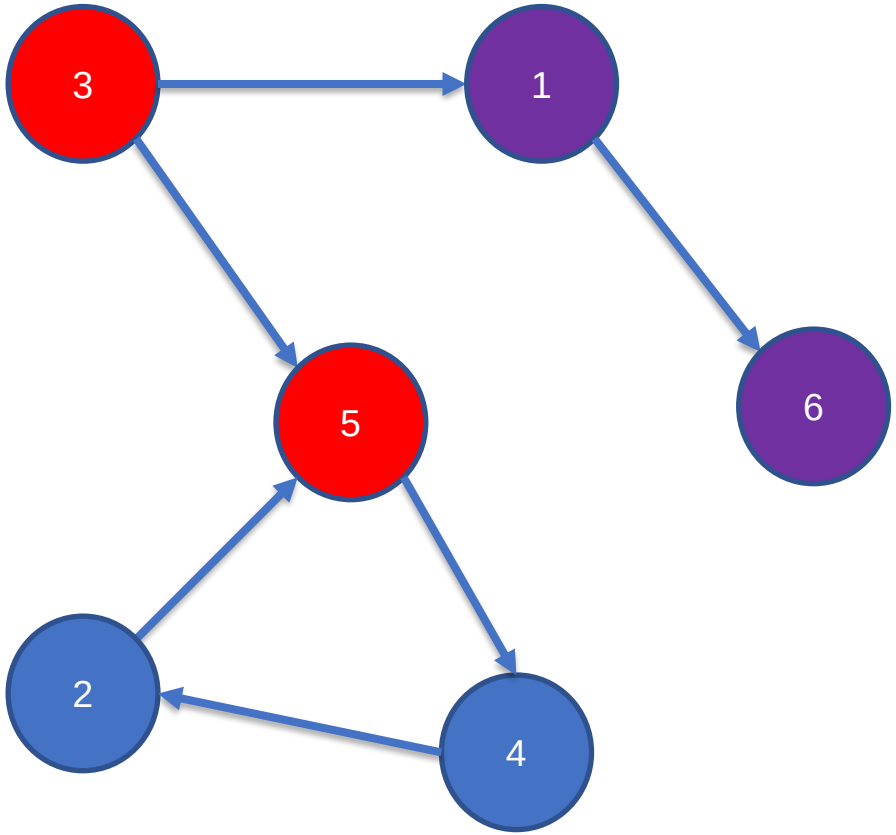


Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	False	False	True	False	False	False
Visited	True	False	True	False	False	True

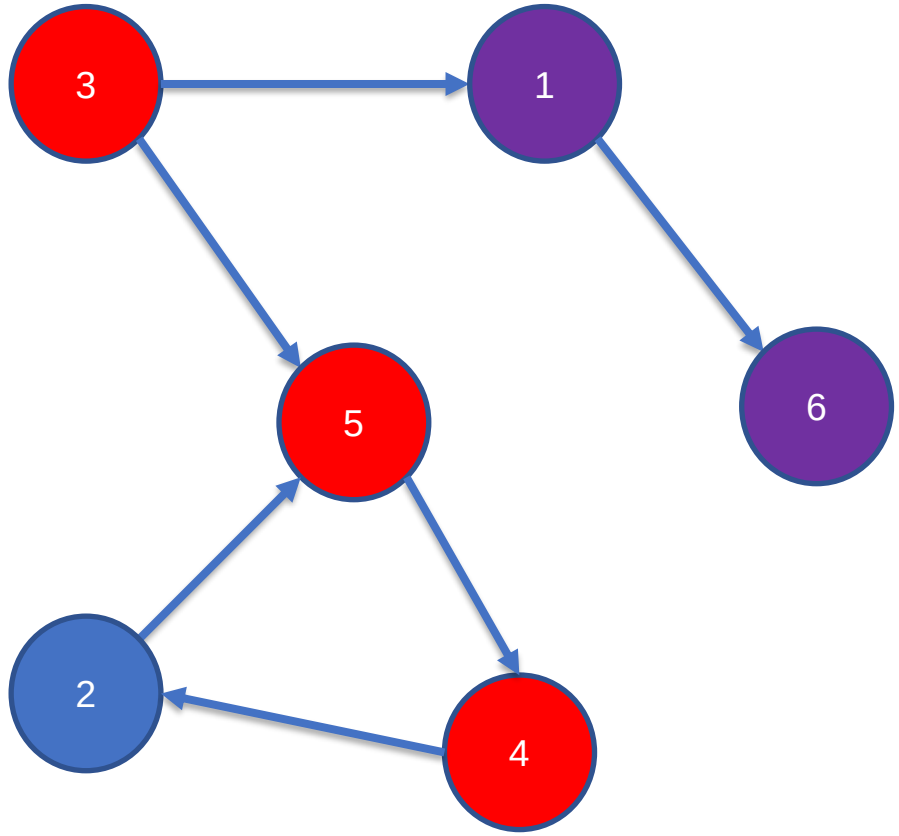
Visited[5] = False nên chúng ta sẽ bắt đầu DFS tiếp tại 5.



Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	False	False	True	False	True	False
Visited	True	False	True	False	True	True



Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	False	False	True	True	True	False
Visited	True	False	True	True	True	True

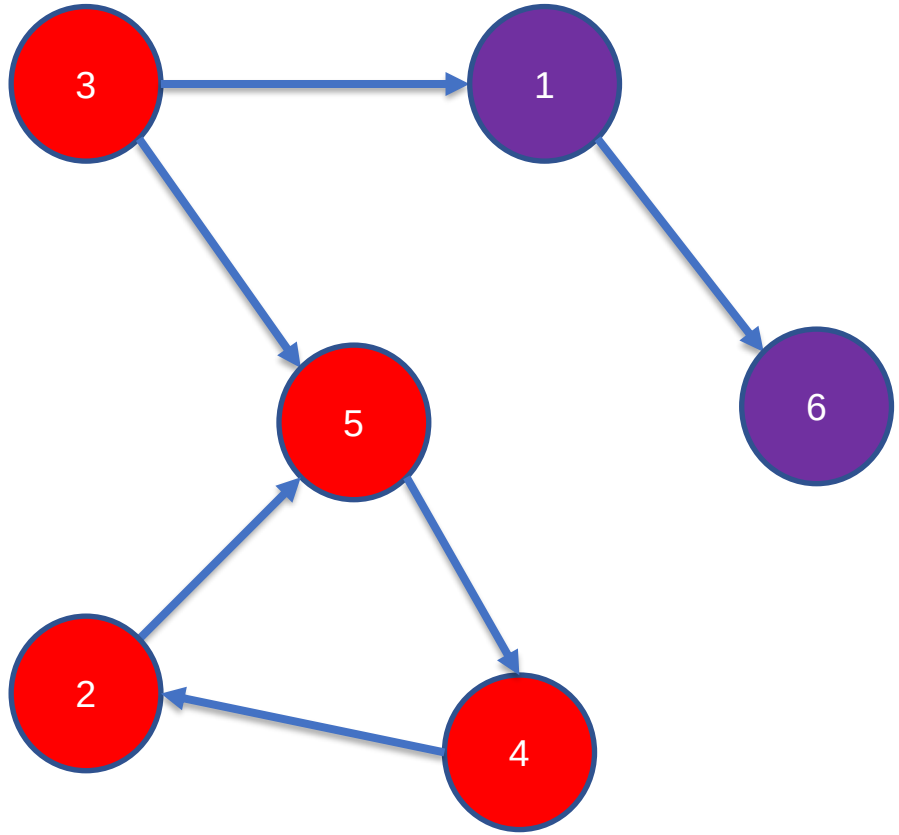


Index	1	2	3	4	5	6
Graph	[6]	[5]	[1, 5]	[2]	[4]	[]
Flag	False	True	True	True	True	False
Visited	True	True	True	True	True	True

Đến đây, do đỉnh 2 có đỉnh kề nó là đỉnh 5 và 5 cũng đang được đánh dấu Flag[5] = True.

➔ Chúng ta đã tìm ra được chu trình trong đồ thị này.

➔ Thuật toán dừng lại và xuất ra đáp án "YES".



Các bước giải

B1: Đọc vào số test T

B2: Đọc vào số đỉnh N và số cạnh M

B3: Khởi tạo mảng Visited, Flag, danh sách kề Graph. Đọc các cạnh vào Graph. Khởi tạo một biến toàn cục answer = "NO".

B4: Cho i chạy từ 1 đến N:

Nếu đỉnh i chưa được duyệt DFS thì DFS(i)

B5: In ra đáp án answer của mỗi test case.

Độ phức tạp: $\boldsymbol{O(T \times (N + M))}$, trong đó N là số đỉnh đề cho và M là số cạnh đề cho.

Mã giả

Mã giả

```
Read(T)
While(T--):
    read(N, M)
    graph = [[], [], [], …]
    flag, Visited = [False, False, False, …]
    for i: 1 to M do
        read(A, B)
        graph[A].append(B)
    answer = “NO”
    for i: 1 to N do
        if (Visited[i] == False):
                DFS(i)
    print(answer)
```

Mã giả

```
Func DFS(start):
    Flag[start] = True
    Visited[start] = True
    for v in graph[start]:
        if (flag[v]): // phải kiểm Flag[v] trước Visited[v]
                answer = “YES”
                return()
        if (Visited[v]):
                continue
        DFS(v)
    Flag[start] = False
    return()
```

Thank you