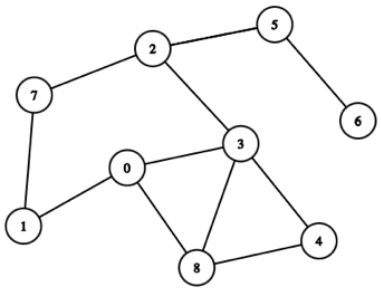
BÀI 3
TÌM KIẾM TRÊN ĐỒ THỊ

Mục tiêu

- Hiểu ý tưởng các thuật toán tìm kiếm trên đồ thị: **tìm kiếm theo chiều sâu** (**D**epth-**F**irst **S**earch) và **tìm kiếm theo chiều rộng** (**B**readth-**F**irst **S**earch);
- Cài đặt được các các thuật toán trên máy tính;
- Vận dụng giải các bài toán liên quan.

1. Một số khái niệm



Hình 1. Đồ thị liên thông

Đường đi (*walk*) từ một đỉnh u đến đỉnh v trong đồ thị $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ là một dãy các đỉnh $\{v_1, v_2, v_3, \dots, v_k\}$ sao cho:

- $v_1 = u$ và $v_k = v$;
- $(v_i, v_{i+1}) \in E$ (với $i = 1, 2, \dots, k-1$)

Ví dụ: Trong Hình 1, một số đường đi từ đỉnh 0 đến đỉnh 6 là: {0, 1, 7, 2, 6}, {0, 3, 2, 5, 6}, {0, 3, 8, 4, 3, 2, 5, 6} (*chú ý mỗi đỉnh có thể được đi qua nhiều lần*)

Đường đi đơn đỉnh (*path*) là đường đi mà mỗi đỉnh chỉ đi qua tối đa một lần. Đường đi đơn đỉnh từ đỉnh đỉnh u đến đỉnh v ký hiệu là $\boldsymbol{P(u, v)}$.

Ví dụ: Trong Hình 1, các đường đi {0, 1, 7, 2, 6}, {0, 3, 2, 5, 6} là các đường đi đơn đỉnh, còn {0, 3, 8, 4, 3, 2, 5, 6} không phải là đường đi đơn đỉnh vì đỉnh 3 được đi qua 2 lần.

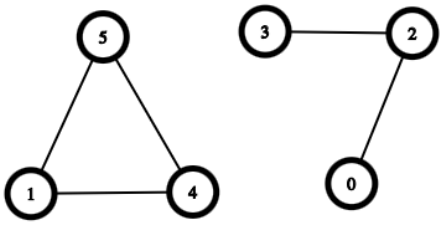
Chu trình (*cycle*) là đường đi đơn đỉnh có đỉnh đầu và đỉnh cuối giống nhau.

Ví dụ: Trong Hình 1, đường đi {0, 1, 7, 2, 3, 0} là một chu trình vì đường đi bắt đầu từ đỉnh 0 và kết thúc cũng tại đỉnh 0.

Đồ thị liên thông (*connected graph*) là đồ thị vô hướng và luôn có đường đi giữa mọi cặp đỉnh. Khi đó ta nói các đỉnh liên thông với nhau.

Ví dụ: Đồ thị trong Hình 1 là đồ thị liên thông vì có đường đi giữa mọi cặp đỉnh trong đồ thị.

Thành phần liên thông (*connected component*) là một tập con các đỉnh trong đồ thị vô hướng, chúng liên thông với nhau và không thể thêm bất kì đỉnh nào khác vào tập này mà vẫn đảm bảo các đỉnh vẫn liên thông.



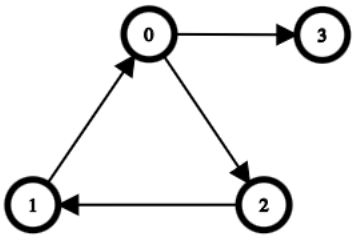
Hình 2. Đồ thị gồm 2 thành phần liên thông

Ví dụ: Đồ thị trong Hình 2 là đồ thị gồm **02 thành phần liên thông**: thành phần thứ nhất gồm các đỉnh {1, 4, 5}, thành phần thứ 2 gồm các đỉnh {0, 2, 3}.

Trên đồ thị có hướng có các khái niệm tương tự:

Đồ thị liên thông mạnh (*strongly connected graph*) là đồ thị có hướng và luôn có đường đi giữa mọi cặp đỉnh.

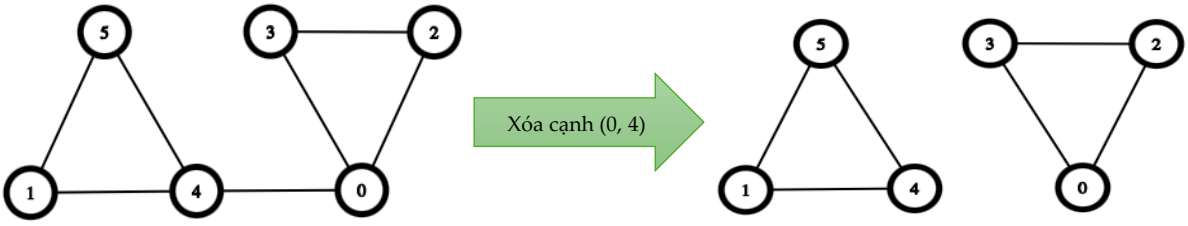
Thành phần liên thông mạnh (*strongly connected component*) là một tập con các đỉnh trong đồ thị có hướng, chúng liên thông với nhau và không thể thêm bất kì đỉnh nào khác vào tập này mà vẫn đảm bảo các đỉnh vẫn liên thông.



Hình 3. Tính liên thông trong đồ thị có hướng

Ví dụ: Đồ thị trong Hình 3 không phải là đồ thị liên thông mạnh vì đỉnh 3 không liên thông với các đỉnh còn lại (*không có đường đi từ đỉnh 3 đến các đỉnh còn lại*). Tuy nhiên, phần đồ thị gồm các đỉnh {0, 1, 2} các cách cạnh liên thuộc với 3 đỉnh đó là một thành phần liên thông mạnh vì có đường đi giữa mọi cặp đỉnh thành phần này.

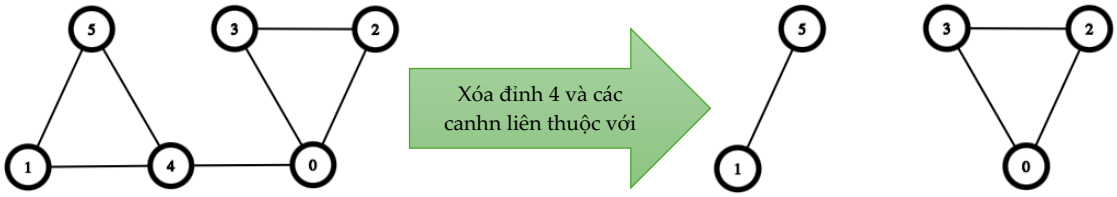
Cạnh cầu (*bridge edge*) là cạnh mà khi xóa cạnh đó sẽ làm tăng số thành phần liên thông của đồ thị.



Hình 4. Cạnh cầu trong đồ thị

Ví dụ: Trong Hình 4,Hình 2 cạnh (0, 4) là cạnh cầu vì khi xóa cạnh (0, 4) số thành phần liên thông của đồ thị tăng từ 1 thành 2.

Đỉnh khớp (*cut vertex*) là đỉnh mà khi xóa bỏ nó và các cạnh liên thuộc với nó sẽ làm tăng số thành phần liên thông của đồ thị.



Hình 5. Đỉnh khớp trong đồ thị

Ví dụ: Trong Hình 5, đỉnh 4 là một đỉnh khớp vì khi xóa đỉnh 4 và các cạnh liên thuộc với nó là (0, 4) (1, 4) và (4, 5) thì số thành phần liên thông của đồ thị tăng từ 1 thành 2. Tương tự, đỉnh 0 cũng là một đỉnh khớp trong đồ thị này.

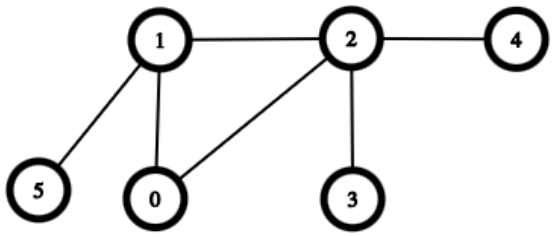
2. Thuật toán tìm kiếm theo chiều sâu (DFS)

2.1. Ý tưởng thuật toán DFS

Thuật toán **Tìm kiếm theo chiều sâu** (**D**epth-**F**irst **S**earch) là thuật toán tìm kiếm trên đồ thị bằng cách xuất phát từ một đỉnh cho trước:

- Tìm và duyệt MỘT ĐỈNH KỀ với đỉnh hiện tại nếu đỉnh đó chưa viếng thăm. Sau đó, xem đỉnh kề vừa tìm được là điểm xuất phát mới và lặp lại quá trình tìm kiếm;
- Nếu không tìm được đỉnh kề nào chưa viếng thăm để đi tiếp, thuật toán sẽ quay lui về đỉnh liền kề trước đó để tìm các hướng đi khác.
- Thuật toán kết thúc khi đã duyệt hết tất cả các hướng đi và không còn đỉnh nào để quay lui được nữa.

Ví dụ. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ nhỏ đến lớn, thứ tự các đỉnh được duyệt bằng thuật toán DFS xuất phát từ đỉnh 1 sẽ là: 1→ 0 → 2 → 3 → 4 → 5.



Giải thích:

- **Xuất phát từ đỉnh 1**: có ba đỉnh kề với đỉnh 1 mà chưa viếng thăm là: 0, 2 và 5. Chọn một đỉnh và ngay lập tức duyệt qua đỉnh này: **duyệt qua đỉnh 0** (theo yêu cầu đề *xét các đỉnh từ nhỏ đến lớn*);
- Tại đỉnh 0: có một đỉnh kề với đỉnh 0 mà chưa viếng thăm là đỉnh 2 nên **duyệt qua đỉnh 2**;
- Tại đỉnh 2: có hai đỉnh kề với đỉnh 2 mà chưa viếng thăm là: 3 và 4. Chọn đỉnh 3 và **duyệt qua đỉnh 3**;
- Tại đỉnh 3: không tìm được bất kỳ đỉnh nào kề với đỉnh 3 mà chưa viếng thăm nên quay lui về đỉnh liền kề trước đỉnh 3 (*đỉnh 2*);
- Tại đỉnh 2: còn một đỉnh kề với đỉnh 2 mà chưa viếng thăm là đỉnh 4 nên **duyệt qua đỉnh 4**;
- Tại đỉnh 4: không tìm được bất kỳ đỉnh nào kề với đỉnh 4 mà chưa viếng thăm nên quay lui về đỉnh liền kề trước đỉnh 4 (*đỉnh 2*);
- Tại đỉnh 2: không tìm được bất kỳ đỉnh nào kề với đỉnh 2 mà chưa viếng thăm nên quay lui về đỉnh liền kề trước đỉnh 2 (*đỉnh 0*);
- Tại đỉnh 0: không tìm được bất kỳ đỉnh nào kề với đỉnh 0 mà chưa viếng thăm nên quay lui về đỉnh liền kề trước đỉnh 0 (*đỉnh 1*);
- Tại đỉnh 1: còn một đỉnh kề với đỉnh 1 mà chưa viếng thăm là đỉnh 5 nên **duyệt qua đỉnh 5**;
- Tại đỉnh 5: không tìm được bất kỳ đỉnh nào kề với đỉnh 0 mà chưa viếng thăm nên quay lui về đỉnh liền kề trước đỉnh 5 (*đỉnh 1*);
- Tại đỉnh 1: không tìm được bất kỳ đỉnh nào kề với đỉnh 1 mà chưa viếng thăm và đây cũng là đỉnh xuất phát đầu tiên nên thuật toán kết thúc.

Do đó, thứ tự các đỉnh đã được duyệt qua bằng thuật toán DFS là:

1→ 0 → 2 → 3 → 4 → 5

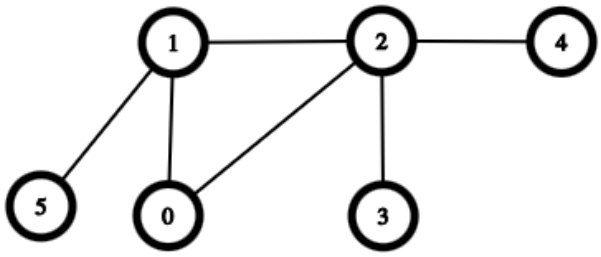
2.2. Cài đặt bằng DFS bằng Đệ quy

Input: đỉnh xuất phát *u*

Output: thứ tự các đỉnh đã duyệt qua bằng thuật toán DFS

```
DFS(u)
   Đánh dấu đã viếng thăm u
   Xử lý đỉnh u
   Xét các đỉnh v kề u
      Nếu đỉnh v chưa được viếng thăm
         DFS(v)
```

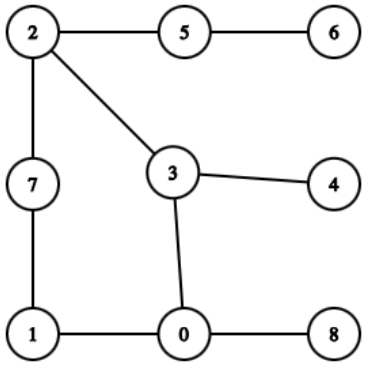
Ví dụ. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ nhỏ đến lớn, minh họa quá trình tìm kiếm trên đồ thị xuất phát từ đỉnh 1 bằng thuật toán DFS (*cài đặt đệ quy*):



Call Stack DFS(u)	**u**	**Các đỉnh đã viếng thăm**	**Đỉnh v kề u và chưa viếng thăm** *(chọn đỉnh nhỏ nhất)*
1	1	1	0
1 → 0	0	1, 0	2
1 → 0 → 2	2	1, 0, 2	3
1 → 0 → 2 → 3	3	1, 0, 2, 3	-
1 → 0 → 2	2	-	4
1 → 0 → 2 → 4	4	1, 0, 2, 3, 4	-
1 → 0 → 2	2	-	-
1 → 0	0	-	-
1	1	-	5
1 → 5	5	1, 0, 2, 3, 4, 5	-
1	1	-	-
-			

Thứ tự các đỉnh được viếng thăm khi duyệt đồ thị bằng thuật toán DFS, xuất phát từ đỉnh 1 là: **1, 0, 2, 3, 4, 5**

Bài tập 1. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ nhỏ đến lớn, minh họa quá trình tìm kiếm trên đồ thị xuất phát từ đỉnh 0 bằng thuật toán DFS (*cài đặt đệ quy*):



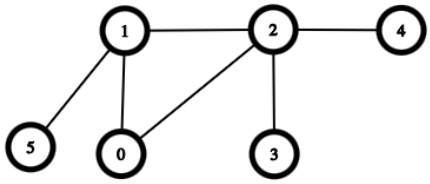
Call Stack DFS(u)	**Đỉnh u**	**Các đỉnh đã viếng thăm**	**Đỉnh v kề u và chưa viếng thăm** *(chọn đỉnh nhỏ nhất)*
0	0	0	1
0 → 1	1	0, 1	7
0 → 1 → 7	7	0,1,7	2
…			
…			

2.3. Cài đặt DFS bằng vòng lặp (sử dụng Stack)

```
DFS(u)
   Thêm u vào đỉnh stack
   Lặp lại cho đến khi stack rỗng
      Lấy u từ đỉnh stack và xóa u khỏi stack
      Nếu u chưa được viếng thăm (một có thể xuất hiện nhiều lần)
         Đánh dấu đã viếng thăm u
         Xử lý đỉnh u
         Xét các đỉnh v kề u
            Nếu đỉnh v chưa được viếng thăm
               Thêm v vào đỉnh stack
```

Lưu ý: *để kết quả thuật toán DFS cài đặt bằng Stack giống với thuật toán DFS cài đặt bằng đệ quy thì phải duyệt danh sách các đỉnh v kề u theo thứ tự đảo ngược.*

Ví dụ. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ lớn đến nhỏ, minh họa quá trình tìm kiếm trên đồ thị xuất phát từ đỉnh 1 bằng thuật toán DFS (*cài đặt bằng Stack*):

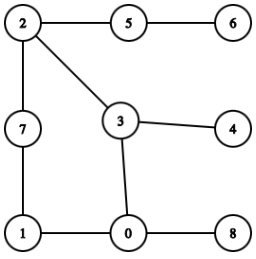


Stack	u	Stack sau khi xóa u	Các đỉnh đã viếng thăm	Stack sau khi thêm các đỉnh v kề u chưa viếng thăm
1	1	-	1	5 → 2 → 0
5 → 2 → 0	0	5 → 2	1, 0	5 → 2 → 2
5 → 2 → 2	2	5 → 2	1, 0, 2	5 → 2 → 4 → 3
5 → 2 → 4 → 3	3	5 → 2 → 4	1, 0, 2, 3	5 → 2 → 4
5 → 2 → 4	4	5 → 2	1, 0, 2, 3, 4	5 → 2
5 → 2	2(*)	5	-	-
5	5	-	1, 0, 2, 3, 4, 5	-
-				

(*) *đỉnh 2 đã được xử lý sau khi duyệt đỉnh 0 nên đỉnh 2 được thêm vào khi duyệt đỉnh 1 sẽ được bỏ qua.*

Thứ tự các đỉnh được viếng thăm khi duyệt đồ thị bằng thuật toán DFS, xuất phát từ đỉnh 1 là: **1, 0, 2, 3, 4, 5**

Bài tập 2. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ lớn đến nhỏ, minh họa quá trình tìm kiếm trên đồ thị xuất phát từ đỉnh 0 bằng thuật toán DFS (*cài đặt bằng Stack*):



Stack	u	Stack sau khi xóa u	Các đỉnh đã viếng thăm	Stack sau khi thêm các đỉnh v kề u chưa viếng thăm
…				
…				

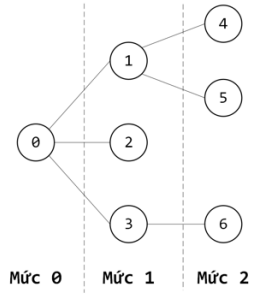
2.4. Độ phức tạp của thuật toán DFS: **O(E + V)**

3. Thuật toán tìm kiếm theo chiều rộng (BFS)

3.1. Ý tưởng thuật toán BFS

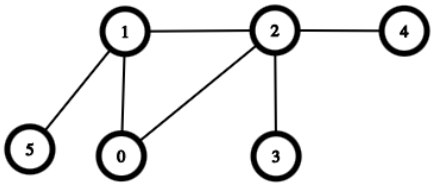
Thuật toán **Tìm kiếm theo chiều rộng** (**B**readth-**F**irst **S**earch) là thuật toán tìm kiếm trên đồ thị bằng cách xuất phát từ một đỉnh cho trước:

- Tìm và duyệt TẤT CẢ ĐỈNH KỀ với đỉnh hiện tại nếu đỉnh đó chưa viếng thăm (*các đỉnh cùng mức*). Sau đó, lần lượt với mỗi đỉnh kề vừa tìm được, thuật toán tiếp tục lan rộng ra các đỉnh kề tiếp theo (*các đỉnh cùng mức phải được xử lý xong trước khi chuyển qua mức tiếp theo*);

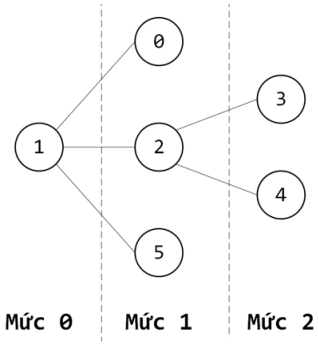


- Thuật toán kết thúc khi đã tất cả các đỉnh ở mức đang xét đều không còn đỉnh kề nào chưa được viếng thăm.

Ví dụ. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ nhỏ đến lớn, thứ tự các đỉnh được duyệt bằng thuật toán BFS xuất phát từ đỉnh 1 sẽ là: 1→ 0 → 2 → 5 → 3 → 4.



Giải thích:



- **Xuất phát từ đỉnh 1**: có ba đỉnh kề với đỉnh 1 mà chưa viếng thăm là: 0, 2 và 5. Lần lượt duyệt các đỉnh này
  → Các đỉnh được duyệt ở mức 1 là: 0, 2, 5;

- Lần lượt xét các đỉnh ở mức 1:
    - Đỉnh 0: có 2 đỉnh kề là: 1, 2 nhưng cả hai đều đã được viếng thăm nên không thể tiếp tục mở rộng từ đỉnh này;
    - Đỉnh 2: có 2 đỉnh kề là: 3, 4 và cả hai đều chưa được viếng thăm nên sẽ duyệt đỉnh này (*mức 2*);
    - Đỉnh 5: kề với đỉnh 1 nhưng đã được viếng thăm nên không thể tiếp tục mở rộng từ đỉnh này;

    → **Các đỉnh được duyệt ở mức 2 là: 3, 4**.
- Lần lượt xét các đỉnh ở mức 2:
    - Đỉnh 0: kề với đỉnh 2 nhưng đã được viếng thăm nên không thể tiếp tục lan rộng từ đỉnh này;
    - Đỉnh 2: kề với đỉnh 2 nhưng đã được viếng thăm nên không thể tiếp tục lan rộng từ đỉnh này;
- Tất cả các đỉnh ở mức 2 đều không còn đỉnh kề nào chưa được viếng thăm nên thuật toán kết thúc.

Do đó, thứ tự các đỉnh đã được duyệt qua bằng thuật toán BFS là:

1→ 0 → 2 → 5 → 3 → 4

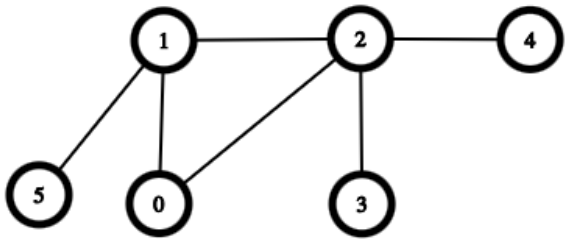
3.2. Cài đặt bằng BFS Queue

Input: đỉnh xuất phát *u*

Output: thứ tự các đỉnh đã duyệt qua bằng thuật toán BFS

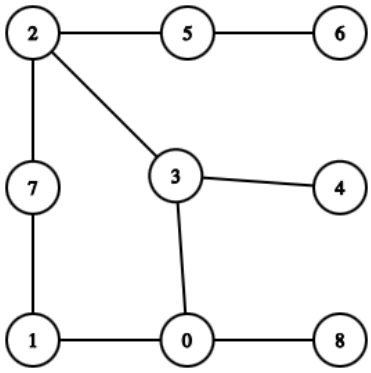
```
BFS(u)
   Thêm u vào đầu queue
   Đánh dấu đã viếng thăm u
   Lặp lại cho đến khi queue rỗng
      Lấy một đỉnh u từ đầu queue
      Xử lý đỉnh u
      Xét các đỉnh v kề u
         Nếu đỉnh v chưa được viếng thăm
            Thêm v vào cuối queue
            Đánh dấu đã viếng thăm v
```

Ví dụ. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ nhỏ đến lớn, minh họa quá trình tìm kiếm trên đồ thị xuất phát từ đỉnh 1 bằng thuật toán BFS:



Queue	Lấy đỉnh u ở đầu queue	Queue sau khi xóa đỉnh u	Các đỉnh đã viếng thăm	Queue sau khi thêm đỉnh v kề u mà chưa viếng thăm vào cuối
1	-	-	1	
1	1	-	1, 0, 2, 5	0 → 2 → 5
0 → 2 → 5	0	2 → 5	-	2 → 5
2 → 5	2	5	1, 0, 2, 5, 3, 4	5 → 3 → 4
5 → 3 → 4	5	3 → 4	-	3 → 4
3 → 4	3	4	-	4
4	4	-	-	-
-				

Bài tập 3. Cho đồ thị như hình vẽ bên dưới. Giả sử xét các đỉnh kề của một đỉnh theo thứ thự từ nhỏ đến lớn, minh họa quá trình tìm kiếm trên đồ thị xuất phát từ đỉnh 0 bằng thuật toán BFS:



Queue	Lấy đỉnh u ở đầu queue	Queue sau khi xóa đỉnh u	Các đỉnh đã viếng thăm	Queue sau khi thêm đỉnh v kề u mà chưa viếng thăm vào cuối
…				
…				

3.3. Độ phức tạp của thuật toán BFS: **O(E + V)**

4. Vận dụng

4.1. Bài toán tìm đường đi

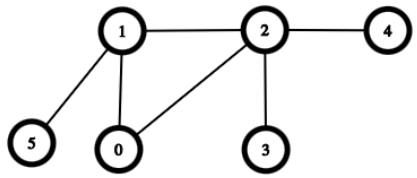
Cho đồ thị $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, đỉnh xuất phát s và đỉnh đến t. Hãy cho biết có đường đi đơn đỉnh từ s đến t hay không? Nếu có hãy chỉ ra một đường đi đơn đỉnh từ s đến t.

Phương pháp:

- Duyệt đồ thị xuất phát từ đỉnh s. Nếu đỉnh t được viếng thăm trong quá trình duyệt đồ thị thì có đường đi từ s đến t;
- Để truy xuất đường đi có thể dùng một mảng để lưu lại đỉnh trước của các đỉnh được viếng thăm (*vì là đường đi đơn đỉnh nên mỗi đỉnh được duyệt chỉ có duy nhất một đỉnh trước nó*).
- Sau khi kết thúc duyệt, xây dựng lại đường đi bằng cách truy ngược từ đỉnh đích về đỉnh bắt đầu.

Ví dụ:

```
DFS(u)
   Đánh dấu đã viếng thăm u
   Xử lý đỉnh u
   Xét các đỉnh v kề u
      Nếu đỉnh v chưa được viếng thăm
         Lưu đỉnh trước của đỉnh v là đỉnh u
         DFS(v)
```



u	0	1	2	3	4	5
pre	1	-1	0	2	2	1

Call Stack DFS(u)	**u**	**Các đỉnh đã viếng thăm**	**Đỉnh v kề u và chưa viếng thăm** *(chọn đỉnh nhỏ nhất)*	**pre[v] = u**
1	1	1	0	pre[0] = 1
$1 \to 0$	0	1, 0	2	pre[2] = 0
$1 \to 0 \to 2$	2	1, 0, 2	3	pre[3] = 2
$1 \to 0 \to 2 \to 3$	3	1, 0, 2, 3	-	
$1 \to 0 \to 2$	2	-	4	pre[4] = 2
$1 \to 0 \to 2 \to 4$	4	1, 0, 2, 3, 4	-	
$1 \to 0 \to 2$	2	-	-	
$1 \to 0$	0	-	-	
1	1	-	5	pre[5] = 1
$1 \to 5$	5	1, 0, 2, 3, 4, 5	-	
1	1	-	-	
-				

4.2. Bài toán xác định các thành phần liên thông

Cho đồ thị $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. Hãy cho đồ thị có phải là đồ thị liên thông hay không? Nếu không thì đồ thị có bao nhiêu thành phần liên thông, liệt kê các đỉnh của từng thành phần liên thông.

Phương pháp:

- Dùng thuật toán DFS/BFS duyệt đồ thị xuất phát từ đỉnh bất kỳ (*thường là đỉnh nhỏ nhất*);
- Nếu có thể từ một đỉnh viếng thăm hết tất cả các đỉnh còn lại của đồ thị thì đồ thị liên thông. Ngược lại, đồ thị không liên thông;
- Trường hợp đồ thị không liên thông, tiếp tục duyệt đồ thị xuất phát từ các đỉnh chưa được viếng cho đến khi tất cả các đỉnh của đồ thị đều được viếng thăm. Số lần gọi hàm DFS/BFS chính là số thành phần liên thông của đồ thị.

4.3. Kiểm tra cạnh cầu, đỉnh khớp

Phương pháp:

- Đếm số thành phần liên thông của đồ thị trước khi xóa cạnh/đỉnh cần kiểm tra;
- Xóa cạnh/đỉnh cần kiểm tra;
- Đếm lại số thành phần liên thông của đồ thị sau khi xóa cạnh/đỉnh cần kiểm tra. Nếu số thành phần liên thông của đồ thị tăng lên thì đó là cạnh cầu/đỉnh khớp.

--- HẾT ---