

Phần I

Lý thuyết đồ thị

Chương 1

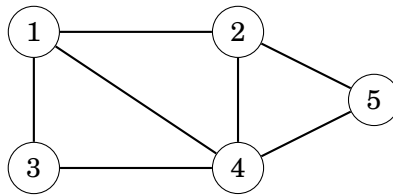
Lý thuyết đồ thị - cơ bản

Các khái niệm về đồ thị và biểu diễn đồ thị.

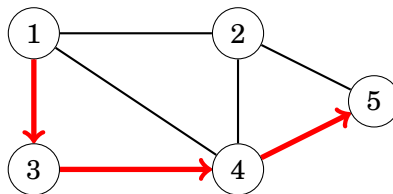
1.1 Các thuật ngữ đồ thị

A **đồ thị** gồm **các đỉnh** và **các cạnh**. Trong các bài toán, thường sẽ là n đỉnh, m cạnh. Các đỉnh được đánh số lần lượt là $1, 2, \dots, n$.

Ví dụ, đồ thị dưới đây gồm 5 đỉnh, 7 cạnh:



Một **đường đi** từ đỉnh a đến đỉnh b thông qua một số cạnh trên đồ thị. Độ dài của đường đi là số cạnh trên đường đi đó. Ví dụ, đồ thị dưới đây có một đường đi là từ đỉnh 1 đến đỉnh 5 là: $1 \rightarrow 3 \rightarrow 4 \rightarrow 5$ với độ dài là 3.

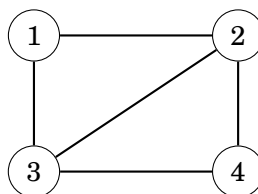


Một đường đi gọi là là **chu trình** nếu đỉnh đầu tiên và đỉnh cuối cùng trùng nhau. Ví dụ, đồ thị trên có một chu trình: $1 \rightarrow 3 \rightarrow 4 \rightarrow 1$. Một đường đi là **đường đi đơn** nếu mỗi nút xuất hiện nhiều nhất một lần trong đường đi.

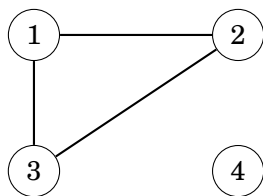
Tính liên thông

Một đồ thị là **liên thông** nếu luôn có một đường đi giữa hai đỉnh bất kỳ trong đồ thị.

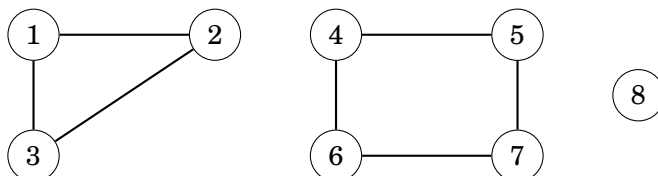
Ví dụ: đồ thị dưới đây là liên thông:



Đồ thị dưới đây không liên thông vì không có đỉnh nào được kết nối với đỉnh 4 của đồ thị:

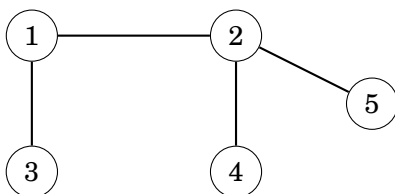


Mỗi phần liên thông trong đồ thị được gọi là **thành phần liên thông**. Ví dụ, đồ thị dưới đây gồm ba thành phần liên thông: {1, 2, 3}, {4, 5, 6, 7} và {8}.



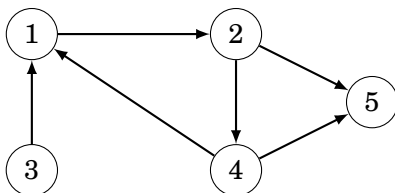
Một **cây đồ thị** là một đồ thị liên thông gồm n đỉnh, $n - 1$ cạnh. Giữa hai đỉnh bất kỳ của cây, chỉ có một đường đi duy nhất giữa hai nút bất kỳ của cây.

Ví dụ, đồ thị dưới đây là một cây:



Cạnh có hướng

Một đồ thị gọi là **có hướng** nếu các cạnh của nó chỉ được duyệt theo một hướng chỉ định. Ví dụ, đồ thị dưới đây là một đồ thị có hướng:

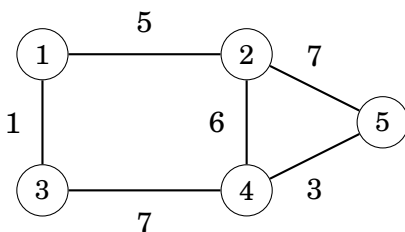


Đồ thị ở trên tồn tại đường đi từ đỉnh 3 tới đỉnh 5: $3 \rightarrow 1 \rightarrow 2 \rightarrow 5$ nhưng không có đường đi nào từ đỉnh 5 tới đỉnh 3.

Cạnh có trọng số

Trong một đồ thị **có trọng số** mỗi cạnh sẽ được gán một giá trị gọi là **trọng số**. Các trọng số thường được hiểu là độ dài cạnh.

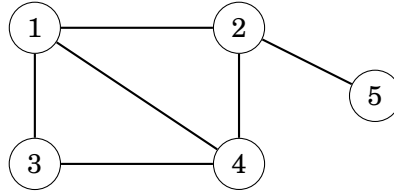
Ví dụ: đồ thị dưới đây là đồ thị có trọng số:



Độ dài của một đường đi trong đồ thị có trọng số là tổng trọng số tất cả các cạnh trên đường đi đó. Ví dụ với đồ thị ở trên, đường đi $1 \rightarrow 2 \rightarrow 5$ có độ dài là 12, còn độ dài của đường đi $1 \rightarrow 3 \rightarrow 4 \rightarrow 5$ là 11, đây cũng là đường đi **ngắn nhất** từ đỉnh 1 đến đỉnh 5.

Đỉnh kề và bậc của đỉnh

Hai đỉnh gọi là **liên kề** nếu tồn tại một cạnh nối trực tiếp giữa chúng. **Bậc** của một đỉnh là số lượng đỉnh kề với đỉnh đó. Ví dụ, đồ thị trên đỉnh 2 có 3 đỉnh liên kề là 1, 4, 5 do đó nó có bậc là 3.


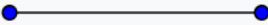
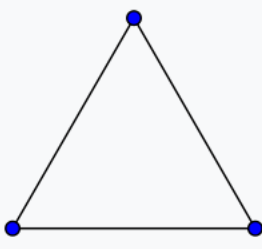
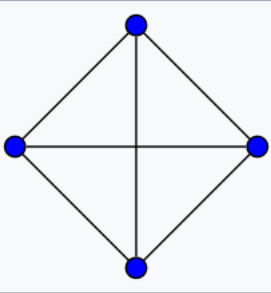
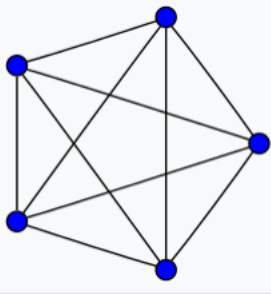
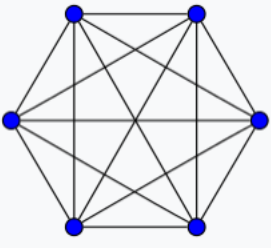
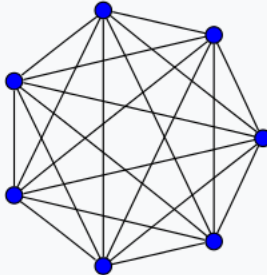
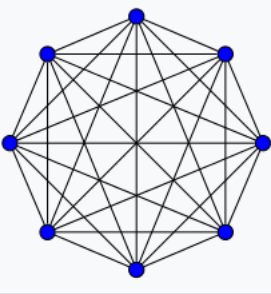
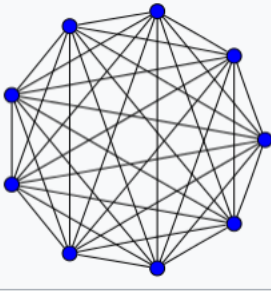
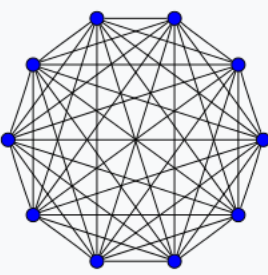
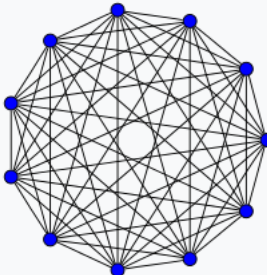
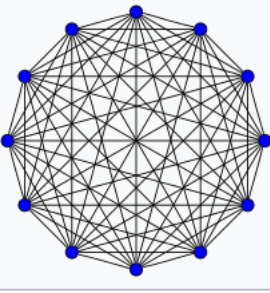


Tổng bậc trong đồ thị luôn là $2 \times m$, với m là số cạnh của đồ thị, bởi vì mỗi cạnh tăng bậc hai đỉnh của nó mỗi nút thêm một. Do đó, tổng bậc của đồ thị luôn chẵn.

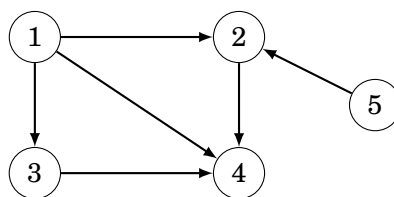
Một đồ thị gọi là **chính quy** nếu các đỉnh đều có cùng bậc. Một đồ thị chính quy với đỉnh có bậc k được gọi là **đồ thị chính quy bậc k** . Đồ thị chính quy bậc 0 gồm các đỉnh không có cạnh chung. Đồ thị chính quy bậc 1 gồm tập các cạnh không nối với nhau, đồ thị chính quy bậc 2 gồm các chu trình không nối với nhau.

Một đồ thị gọi là **đầy đủ** mỗi đỉnh đều có bậc là $n - 1$. Trong đồ thị đầy đủ, luôn có cạnh nối trực tiếp hai đỉnh bất kỳ. Đồ thị đầy đủ là đồ thị đơn có nhiều cạnh nhất và là đồ thị chính quy bậc $n - 1$.

Dưới đây là hình minh họa một số đồ thị đầy đủ với số đỉnh từ 1 đến 12:

$K_1 : 0$	$K_2 : 1$	$K_3 : 3$	$K_4 : 6$
			
$K_5 : 10$	$K_6 : 15$	$K_7 : 21$	$K_8 : 28$
			
$K_9 : 36$	$K_{10} : 45$	$K_{11} : 55$	$K_{12} : 66$
			

Trong đồ thị có hướng, **bậc trong** của một đỉnh là số cạnh kết thúc tại đỉnh đó và **bậc ngoài** của một đỉnh là số cạnh bắt đầu tại đỉnh đó. Ví dụ, trong biểu đồ sau, bậc trong của nút 2 là 2 và bậc ngoài của nút 2 là 1.

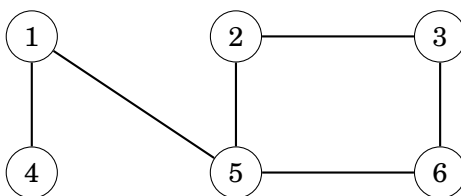


Tô màu

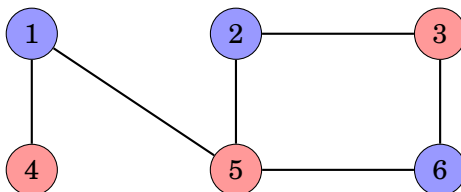
Khi **tô màu** cho một đồ thị, mỗi đỉnh sẽ được gán cho một màu sao cho không có hai đỉnh kề nhau nào được tô cùng một màu.

Một đồ thị gọi là **đồ thị hai phía** nếu có thể tô màu nó bằng hai màu. Như vậy, một đồ thị hai phía sẽ không có chu trình với số cạnh lẻ.

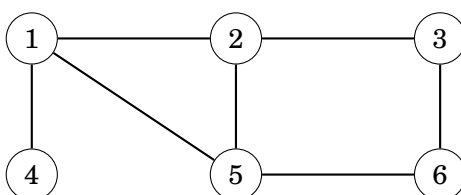
Ví dụ, đồ thị



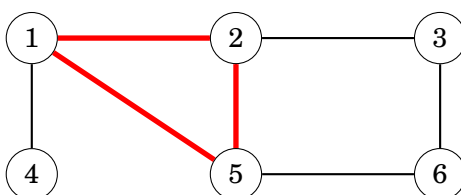
là đồ thị hai phía vì có thể tô nó bằng hai màu như sau:



Tuy nhiên, đồ thị

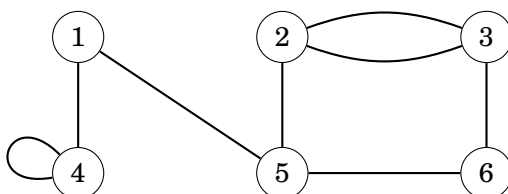


không phải là đồ thị hai phía vì không thể tô màu cho chu trình ba đỉnh bằng cách sử dụng hai màu:



Đồ thị đơn giản

Một đồ thị gọi là **đơn giản** nếu không có cạnh nào xuất phát và kết thúc tại cùng một đỉnh và không có nhiều cạnh nối giữa hai đỉnh. Trong các bài toán, ta thường giả sử đồ thị là đơn giản. Ví dụ đồ thị dưới đây không phải là đồ thị đơn giản:



1.2 Biểu diễn đồ thị

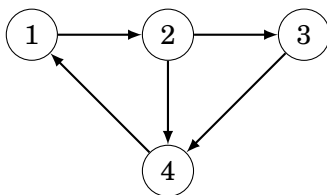
Có một số cách để biểu diễn đồ thị trong thuật toán. Việc lựa chọn cấu trúc dữ liệu phụ thuộc vào kích thước của đồ thị và cách xử lý đồ thị của thuật toán.

Biểu diễn bằng danh sách kề

Trong biểu diễn danh sách kề, mỗi nút x trong đồ thị được gán một **danh sách kề** bao gồm các nút có cạnh nối từ đỉnh x . Danh sách kề là cách phổ biến nhất để biểu diễn đồ thị và hầu hết các thuật toán có thể được triển khai hiệu quả bằng cách sử dụng nó. Một cách thuận tiện để lưu trữ danh sách kề là khai báo một mảng vectơ như sau:

```
vector<int> adj[N];
```

Với hằng số N là số đỉnh tối đa của đồ thị. Ví dụ, đồ thị



có thể được lưu trữ như sau:

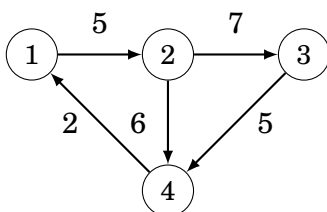
```
adj[1].push_back(2);  
adj[2].push_back(3);  
adj[2].push_back(4);  
adj[3].push_back(4);  
adj[4].push_back(1);
```

Nếu đồ thị vô hướng thì vẫn lưu trữ tương tự có điều phải đưa vào danh sách kề của cả hai đỉnh (lúc này có thể coi như là hai hướng).

Với đồ thị có trọng số thì có thể lưu như sau:

```
vector<pair<int,int>> adj[N];
```

Khi đó, danh sách các cạnh kề với đỉnh a sẽ chứa các cặp (b, w) ứng với một cạnh nối từ a đến b có trọng số là w . Ví dụ, đồ thị



Có thể lưu trữ như sau:

```
adj[1].push_back({2,5});  
adj[2].push_back({3,7});  
adj[2].push_back({4,6});  
adj[3].push_back({4,5});  
adj[4].push_back({1,2});
```

Với việc sử dụng các biểu diễn bằng danh sách kề, ta có thể tìm các đỉnh hiệu quả trong quá trình duyệt đường đi trên đồ thị. Ví dụ, vòng lặp sau sẽ duyệt qua tất cả các đỉnh kề với đỉnh s .

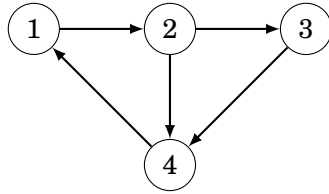
```
for (auto u : adj[s]) {  
    // xử lý đỉnh u kề với s  
}
```

Biểu diễn đồ thị bằng ma trận kề

ma trận kề là một mảng hai chiều cho biết đồ thị chứa các cạnh nào. Cách biểu diễn này cho phép tìm cạnh nối giữa hai đỉnh một cách hiệu quả. Ma trận có thể được lưu trữ dưới dạng một mảng


```
int adj[N][N];
```

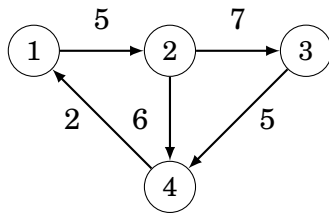
trong đó mỗi giá trị $adj[a][b]$ cho biết đồ thị có tồn tại cạnh nối hai đỉnh a và b hay không. Nếu tồn tại thì $adj[a][b]=1$, còn nếu không thì $adj[a][b]=0$. Ví dụ, đồ thị



có thể được biểu diễn như sau:

	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	1	0	0	0

Nếu đồ thị có trọng số, giá trị của $adj[a][b]$ được gán bằng trọng số của cạnh nối hai đỉnh a và b . Ví dụ, đồ thị



tương ứng với ma trận sau:

	1	2	3	4
1	0	5	0	0
2	0	0	7	6
3	0	0	0	5
4	2	0	0	0

Nhược điểm của phương pháp biểu diễn đồ thị bằng ma trận kề là ma trận gồm 2^n phần tử trong đó hầu hết đều có giá trị bằng 0. Do đó, cách biểu diễn này thường không được sử dụng trong các đồ thị có kích thước lớn.

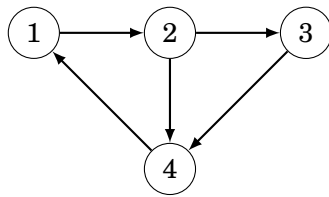
Biểu diễn đồ thị bằng danh sách cạnh

Một **danh sách cạnh** chứa tất cả các cạnh của đồ thị. Cách biểu diễn này thuận tiện trong trường hợp thuật toán chỉ xử lý các cạnh của đồ thị mà không cần xác định cạnh bắt đầu từ một nút nào đó.

Danh sách kề có thể lưu trữ dưới dạng một vector

```
vector<pair<int,int>> edges;
```

Với mỗi cặp (a, b) xác định một cạnh nối từ đỉnh a đến đỉnh b .
Theo đó, đồ thị



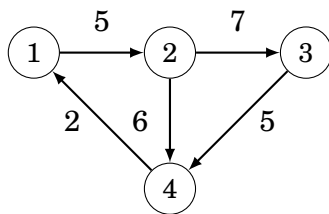
có thể biểu diễn như sau:

```
edges.push_back({1, 2});  
edges.push_back({2, 3});  
edges.push_back({2, 4});  
edges.push_back({3, 4});  
edges.push_back({4, 1});
```

Nếu đồ thị có trọng số thì có thể được lưu như sau:

```
vector<tuple<int, int, int>> edges;
```

Mỗi phần tử trong danh sách có dạng (a, b, w) , mô tả một cạnh nối từ đỉnh a đến đỉnh b có trọng số là w . Ví dụ, đồ thị



có thể biểu diễn như sau ¹:

```
edges.push_back({1, 2, 5});  
edges.push_back({2, 3, 7});  
edges.push_back({2, 4, 6});  
edges.push_back({3, 4, 5});  
edges.push_back({4, 1, 2});
```

¹trong một số trình biên dịch cũ, hàm `make_tuple` phải được sử dụng thay vì dùng cặp ngoặc nhọn (ví dụ dùng `make_tuple(1, 2, 5)` thay vì dùng `{1, 2, 5}`).

Chương 2

Duyệt đồ thị

Chương này thảo luận về hai thuật toán đồ thị cơ bản: tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng. Cả hai thuật toán đều bắt đầu từ một đỉnh trong đồ thị và chúng truy cập tất cả các đỉnh có thể đến được từ đỉnh đó. Sự khác biệt trong hai thuật toán này là thứ tự chúng truy cập các đỉnh.

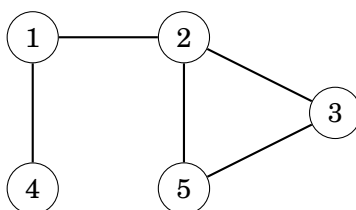
2.1 Tìm kiếm theo chiều sâu - DFS

Tìm kiếm theo chiều sâu DFS là một kỹ thuật duyệt đồ thị đơn giản. Thuật toán bắt đầu từ một đỉnh và duyệt đến tất cả các đỉnh khác có thể truy cập được từ đỉnh này bằng cách sử dụng các cạnh trong đồ thị.

DFS luôn duyệt theo một đường đi duy nhất trong đồ thị miễn là nó tìm thấy các đỉnh mới. Sau đó, nó quay trở lại các đỉnh trước đó và bắt đầu duyệt tiếp các phần khác của đồ thị. Thuật toán theo dõi các đỉnh đã truy cập để nó chỉ xử lý mỗi đỉnh một lần.

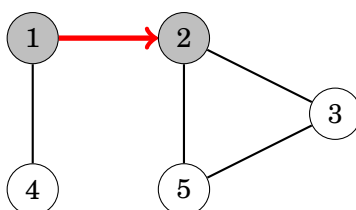
Ví dụ

Hãy quan sát cách mà **DFS** hoạt động trên đồ thị sau:

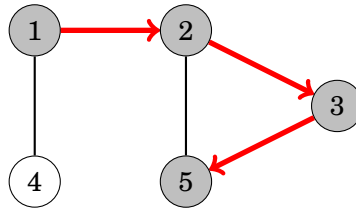


Ta có thể bắt đầu duyệt từ bất kỳ đỉnh nào của đồ thị; giả sử bắt đầu tại đỉnh 1.

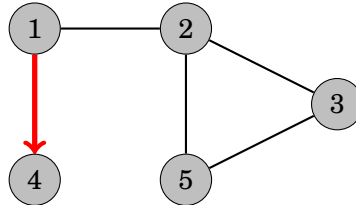
Đầu tiên sẽ duyệt đến đỉnh 2:



Sau đó duyệt đến đỉnh 3 rồi đến đỉnh 5:



Các đỉnh kề của 5 là 2 và 3, nhưng vì cả hai đỉnh này đều đã được duyệt nên lúc này thuật toán sẽ quay lại các đỉnh trước đó. Ngoài ra, các đỉnh kề của 3 và 2 cũng đều duyệt rồi nên tiếp theo sẽ đi từ đỉnh 1 sang đỉnh 4.



Sau đó, quá trình tìm kiếm kết thúc vì tất cả các đỉnh đã được duyệt.

Độ phức tạp về thời gian của tìm kiếm theo chiều sâu là $O(n + m)$ trong đó n là số đỉnh còn m là số cạnh, bởi vì thuật toán xử lý mỗi đỉnh và mỗi cạnh cạnh một lần.

Cài đặt thuật toán

Tìm kiếm theo chiều sâu có thể cài đặt thuận tiện bằng cách sử dụng hàm đệ quy. Hàm `dfs` sau bắt đầu tìm kiếm theo chiều sâu tại một đỉnh nhất định. Hàm này giả định rằng đồ thị được lưu trữ dưới dạng danh sách kề

```
vector<int> adj[N];
```

và đồng thời duy trì một mảng

```
bool visited[N];
```

để theo dõi xem các đỉnh đã được truy cập hay chưa. Ban đầu, tất cả các giá trị của mảng này bằng `false`, và khi duyệt đến đỉnh `s`, thì gán `visited[s]` bằng `true`.

Hàm này có thể triển khai như sau:

```
void dfs(int s) {
    if (visited[s]) return;
    //neu dinh s da tham thi khong tham nua
    visited[s] = true; //danh dau tham dinh s
    // xu ly dinh s
    for (auto u: adj[s]) { //duyet tat ca cac dinh ke voi s
        dfs(u); //goi de quy tham dinh u
    }
}
```

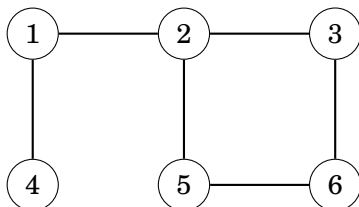
2.2 Tìm kiếm theo chiều rộng

Tìm kiếm theo chiều rộng duyệt các đỉnh theo thứ tự tăng dần của khoảng cách tới đỉnh xuất phát. Do đó, ta có thể tính toán khoảng cách từ đỉnh xuất phát đến tất cả các đỉnh khác bằng cách sử dụng **BFS**. Tuy nhiên, **BFS** khó cài đặt hơn so với **DFS**.

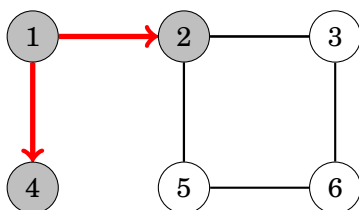
Tìm kiếm theo chiều rộng đi qua các đỉnh từ cấp này đến cấp khác. Đầu tiên duyệt các đỉnh có khoảng cách từ đỉnh xuất phát là 1, sau đó là các đỉnh có khoảng cách là 2, v.v. Quá trình này tiếp tục cho đến khi tất cả các đỉnh đều được duyệt.

Ví dụ

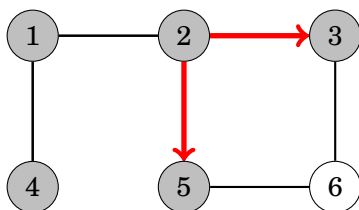
Xét cách mà **BFS** hoạt động trên đồ thị sau:



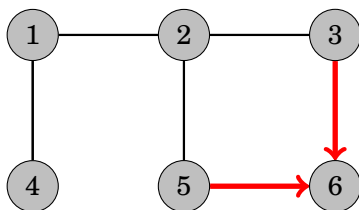
Giả sử tìm kiếm bắt đầu từ đỉnh 1. Đầu tiên, ta sẽ xử lý tất cả các đỉnh có thể đến được từ đỉnh 1 mà chỉ duyệt qua đúng 1 cạnh:



Sau đó, ta xử lý đến đỉnh 3 và đỉnh 5:



Cuối cùng, thăm nốt đỉnh 6:



Bây giờ ta tính toán khoảng cách từ đỉnh xuất phát đến tất cả các đỉnh của đồ thị:

node	distance
1	0
2	1
3	2
4	1
5	2
6	3

Giống như tìm kiếm theo chiều sâu, độ phức tạp của thuật toán **BFS** cũng là $O(n + m)$, với n là số đỉnh còn m là số cạnh của đồ thị.

Cài đặt thuật toán

Tìm kiếm theo chiều rộng khó cài đặt hơn so với tìm kiếm theo chiều sâu, bởi vì thuật toán truy cập các đỉnh thuộc các phần khác nhau của đồ thị. Mã cài đặt điển hình dựa trên cấu trúc hàng đợi có chứa các đỉnh. Tại mỗi bước, đỉnh tiếp theo trong hàng đợi sẽ được xử lý.

Dưới đây là mã cài đặt giả sử đồ thị được lưu trữ dưới dạng danh sách kề và duy trì các thông tin sau:

```
queue<int> q;  
bool visited[N];  
int distance[N];
```

Hàng đợi q chứa các đỉnh được xử lý theo thứ tự tăng dần về khoảng cách so với đỉnh xuất phát. Các đỉnh mới luôn được thêm vào cuối hàng đợi và đỉnh ở đầu hàng đợi là đỉnh được xử lý tiếp theo. Mảng `visited` cho biết một đỉnh đã được truy cập hay chưa còn mảng `distance` sẽ chứa khoảng cách từ đỉnh xuất phát đến tất cả các đỉnh của đồ thị.

Việc tìm kiếm xuất phát từ đỉnh x có thể được cài đặt như sau:

```
visited[x] = true;  
distance[x] = 0;  
q.push(x);  
while (!q.empty()) { //xu ly tat ca cac dinh trong hang doi  
    int s = q.front(); q.pop(); //lay dinh s o dau hang doi  
    // xu ly dinh s  
    for (auto u : adj[s]) { //duyet tat ca cac dinh ke voi dinh s  
        if (visited[u]) continue;  
        //neu dinh dang xet da duoc truy cap thi bo qua  
        visited[u] = true;  
        distance[u] = distance[s]+1;  
        q.push(u); //dua dinh ke voi s nay vao cuoi hang doi  
    }  
}
```

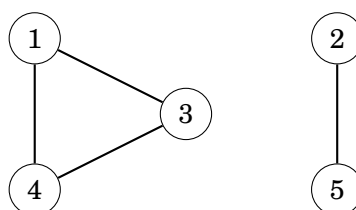
2.3 Ứng dụng của các thuật toán duyệt đồ thị

Sử dụng thuật toán duyệt đồ thị, chúng ta có thể kiểm tra nhiều tính chất của đồ thị. Thông thường, cả tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng đều có thể được sử dụng, nhưng trong thực tế, tìm kiếm theo chiều sâu là lựa chọn tốt hơn vì nó dễ cài đặt hơn. Trong các ứng dụng sau đây, chúng ta sẽ giả sử rằng đồ thị là vô hướng.

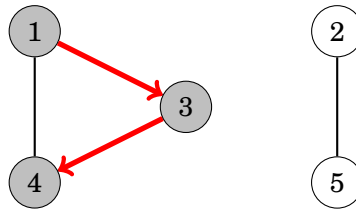
Kiểm tra tính liên thông

Một đồ thị là liên thông nếu luôn có một đường đi giữa hai đỉnh bất kỳ của đồ thị. Do đó, chúng ta có thể kiểm tra xem một đồ thị có liên thông hay không bằng cách bắt đầu tại một đỉnh tùy ý và tìm hiểu xem liệu có thể đến được tất cả các đỉnh khác hay không.

Ví dụ, trong đồ thị



tìm kiếm theo chiều sâu xuất phát từ đỉnh 1 sẽ thăm các đỉnh như sau:

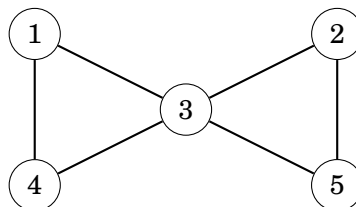


Vì sau khi tìm kiếm không truy cập hết tất cả các đỉnh, ta có thể kết luận rằng đồ thị không liên thông. Theo cách tương tự, chúng ta cũng có thể tìm thấy tất cả các thành phần liên thông của đồ thị bằng cách **DFS** qua tất cả các đỉnh chưa thuộc bất kỳ một thành phần liên thông nào.

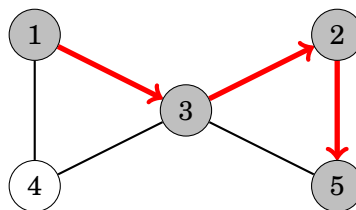
Tìm kiếm chu trình

Một đồ thị chứa một chu trình nếu trong quá trình duyệt qua đồ thị, ta tìm thấy một đỉnh có đỉnh kề (không phải là đỉnh vừa duyệt trước đó trong đường đi hiện tại) đã được thăm.

Ví dụ, đồ thị



có hai chu trình và ta có thể tìm được một chu trình trong đó như sau:



Sau khi di chuyển từ đỉnh 2 đến đỉnh 5, ta thấy rằng đỉnh kề 3 của đỉnh 5 đã được thăm. Như vậy, đồ thị chứa một chu trình đi qua đỉnh 3, chẳng hạn $3 \rightarrow 2 \rightarrow 5 \rightarrow 3$.

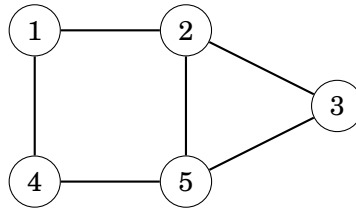
Một cách khác để biết liệu một đồ thị có chứa chu trình hay không là chỉ cần tính số đỉnh và cạnh trong mỗi thành phần liên thông. Nếu một thành phần liên thông có c đỉnh và không có chu trình, thì nó phải chứa chính xác $c - 1$ cạnh (vì vậy nó phải là một cây). Nếu có c hoặc nhiều cạnh hơn, thành phần đó chắc chắn chứa một chu trình.

Kiểm tra đồ thị hai phía

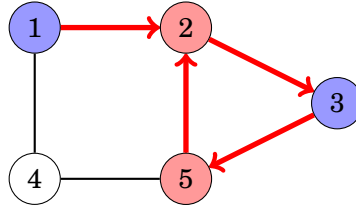
Một đồ thị là đồ thị hai phía nếu các đỉnh của nó có thể được tô bằng hai màu sao cho không có hai đỉnh liên tiếp nào có cùng màu. Có thể dễ dàng kiểm tra xem một đồ thị có phải là hai phía hay không bằng cách sử dụng thuật toán duyệt đồ thị.

Ý tưởng là tô màu đỉnh bắt đầu màu xanh, tất cả nút kề nó màu đỏ, tất cả nút kề tiếp theo màu xanh, v.v. Nếu trong quá trình tìm kiếm mà nhận thấy hai đỉnh liên tiếp cùng màu, có nghĩa là đồ thị không phải là đồ thị hai phía. Ngược lại, đồ thị là hai phía.

Ví dụ, đồ thị



Không phải là đồ thị hai phía vì khi xuất phát từ đỉnh 1, quá trình tô màu như sau:



Có thể thấy rằng màu của cả hai đỉnh 2 và 5 đều là màu đỏ, và hai đỉnh này là liền kề. Do đó, đồ thị không phải là đồ thị hai phía.

Thuật toán này luôn hoạt động, bởi vì khi chỉ có hai màu, màu của đỉnh xuất phát trong một thành phần liên thông sẽ xác định màu của tất cả các nút khác trong thành phần liên thông. Đỉnh xuất phát màu đỏ hay xanh không làm thay đổi kết quả.

Lưu ý rằng trong trường hợp chung, rất khó để biết rằng liệu các đỉnh trong đồ thị có thể được tô màu bằng cách sử dụng k màu sao cho không có đỉnh liền kề nào có cùng màu hay không. Ngay cả khi $k = 3$, không có thuật toán hiệu quả nào được làm được điều đó nhưng đây là vấn đề **NP-hard**.

2.4 Bài tập vận dụng

2.4.1 Kiểm tra cây

Cho đồ thị vô hướng không trọng số gồm n đỉnh, m cạnh.

Hãy cho biết đồ thị đã cho có phải là một cây hay không.

Dữ liệu

- Dòng 1: ghi hai số nguyên n, m ($0 \leq n \leq 10^4, 0 \leq m \leq 2 \times 10^4$);
- Tiếp theo mà m dòng, mỗi dòng ghi hai số nguyên u, v ($1 \leq u, v \leq n$) mô tả một cạnh nối giữa hai đỉnh u, v .

Kết quả

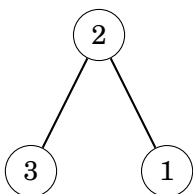
- Ghi “YES” nếu đồ thị đã cho là một cây. Ngược lại ghi “NO”.

Ví dụ

input	output
3 2 1 2 2 3	YES

Giải thích ví dụ

- Hình mô tả đồ thị trong ví dụ:



Gợi ý

- Một đồ thị n đỉnh là một cây nếu thỏa mãn các điều kiện sau:
 1. Đồ thị gồm đúng $n - 1$ cạnh;
 2. Đồ thị không có chu trình;
 3. Đồ thị liên thông.

2.4.2 ConẾch

Một conẾch đang ở toạ độ 1 trên trục toạ độ Ox và cần nhảy đến toạ độ n . Mỗi lần nhảy, nó có thể nhảy sang phải từ 1 đến không quá d đơn vị. Thêm vào đó,Ếch chỉ có thể nhảy lên các vị trí có lá sen.

Hãy lập trình tính xem conẾch cần nhảy ít nhất bao nhiêu lần để tới được toạ độ n .

Dữ liệu

- Dòng 1: ghi hai số nguyên n và d ($2 \leq n \leq 100, 1 \leq d \leq n - 1$)
- Dòng 2: ghi một xâu gồm n kí tự '0' và '1'. Với kí tự thứ i ($1 \leq i \leq n$) bằng '1' xác định toạ độ thứ i có lá sen và ngược lại. Dữ liệu đảm bảo toạ độ 1 và n luôn có lá sen.

Kết quả

- Ghi một số nguyên duy nhất là số lần nhảy ít nhất để conẾch nhảy được đến vị trí n . Nếu không có cách nhảy tới vị trí n thì ghi '-1'.

Ví dụ

input	output
8 4 10010101	2
4 2 1001	-1

Giải thích ví dụ

- Trong ví dụ thứ nhất,Ếch nhảy như sau: $1 \rightarrow 4 \rightarrow 8$.
- Trong ví dụ thứ hai,Ếch không thể nhảy đến toạ độ 4 được vì nó chỉ có thể nhảy tối đa 2 bước sang phải.

2.4.3 Đếm đường đi đơn

Cho đồ thị vô hướng không trọng số gồm n đỉnh, m cạnh. Các đỉnh và các cạnh đều được đánh số từ 1. Mỗi đỉnh của đồ thị có bậc không quá 10.

Gọi K là số lượng đường đi đơn (đường đi không có cạnh lặp lại). Hãy in ra giá trị: $\min(K, 10^6)$.

Dữ liệu

- Dòng 1: ghi hai số nguyên n và m ($1 \leq n \leq 2 \times 10^5, 0 \leq m \leq \min(2 \times 10^5, \frac{n(n-1)}{2})$) tương ứng với số đỉnh và số cạnh của đồ thị.
- Tiếp theo là m dòng, dòng thứ i ghi hai số nguyên u_i, v_i ($1 \leq i \leq m$) mô tả cạnh thứ i nối hai đỉnh u_i và v_i .

Kết quả

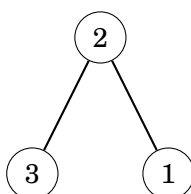
- Ghi một số nguyên duy nhất là $\min(K, 10^6)$ tìm được.

Ví dụ

input	output
4 2 1 2 2 3	3
8 21 2 6 1 3 5 6 3 8 3 6 4 7 4 6 3 4 1 5 2 4 1 2 2 7 1 4 3 5 2 5 2 3 4 5 3 7 6 7 5 7 2 8	2023

Giải thích ví dụ

- Trong ví dụ thứ nhất, có ba đường đi là: $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$. Lưu ý rằng đường đi chỉ gồm một đỉnh (độ dài bằng 0) cũng được đếm.



2.4.4 Đếm thành phần liên thông

Cho đồ thị vô hướng không trọng số gồm n đỉnh, m cạnh. Các đỉnh và các cạnh đều được đánh số từ 1.

Hãy đếm xem trong đồ thị có bao nhiêu thành phần liên thông.

Dữ liệu

- Dòng 1: ghi hai số nguyên n và m ($1 \leq n \leq 100, 0 \leq m \leq \frac{n(n-1)}{2}$) tương ứng với số đỉnh và số cạnh của đồ thị.
- Tiếp theo là m dòng, dòng thứ i ghi hai số nguyên u_i, v_i ($1 \leq i \leq m$) mô tả cạnh thứ i nối hai đỉnh u_i và v_i .

Kết quả

- Ghi một số nguyên duy nhất là số lượng thành phần liên thông của đồ thị.

Ví dụ

input	output
5 3 1 2 1 3 4 5	2
5 0	5
4 6 1 2 1 3 1 4 2 3 2 4 3 4	1

Giải thích ví dụ 1

