

# Chương 1. Các thuật toán tìm kiếm trên đồ thị

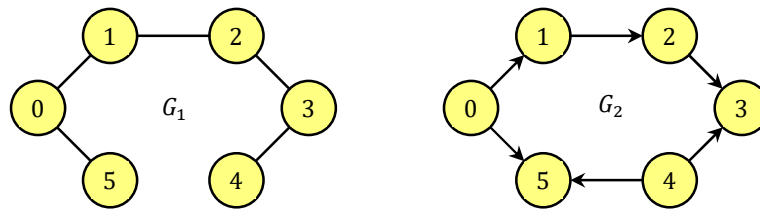
## 1.1. Bài toán tìm đường

Cho đồ thị  $G = (V, E)$  và hai đỉnh  $s, t \in V$ .

Nhắc lại định nghĩa đường đi: Một dãy các đỉnh:

$$P = \langle s = p_0, p_1, \dots, p_k = t \rangle, (\forall i: (p_{i-1}, p_i) \in E)$$

được gọi là một đường đi từ  $s$  tới  $t$ , đường đi này gồm  $k + 1$  đỉnh  $p_0, p_1, \dots, p_k$  và  $k$  cạnh  $(p_0, p_1), (p_1, p_2), \dots, (p_{k-1}, p_k)$ . Đỉnh  $s$  được gọi là đỉnh đầu và đỉnh  $t$  được gọi là đỉnh cuối của đường đi. Nếu tồn tại một đường đi từ  $s$  tới  $t$ , ta nói  $s$  đến được  $t$  và  $t$  đến được từ  $s$ :  $s \leadsto t$ .



Hình 1-1. Đồ thị và đường đi

Trên cả hai đồ thị ở Hình 1-1,  $\langle 0,1,2,3 \rangle$  là đường đi từ đỉnh 0 tới đỉnh 3.  $\langle 0,5,4,3 \rangle$  không phải đường đi vì không có cạnh (cung)  $(5,4)$ .

Một bài toán quan trọng trong lý thuyết đồ thị là bài toán duyệt tất cả các đỉnh có thể đến được từ một đỉnh xuất phát nào đó. Vấn đề này đưa về một bài toán liệt kê mà yêu cầu của nó là không được bỏ sót hay lặp lại bất kỳ đỉnh nào. Chính vì vậy mà ta phải xây dựng những thuật toán cho phép duyệt một cách hệ thống các đỉnh, những thuật toán như vậy gọi là những thuật toán tìm kiếm trên đồ thị (*graph traversal*). Ta quan tâm đến hai thuật toán cơ bản nhất: thuật toán tìm kiếm theo chiều sâu và thuật toán tìm kiếm theo chiều rộng.

Khuôn dạng Input/Output quy định như sau:

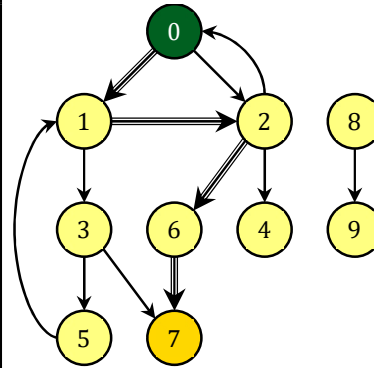
### Input

- ✦ Dòng 1 chứa 4 số  $n, m, s, t$  lần lượt là số đỉnh, số cung, đỉnh xuất phát và đỉnh cần đến của một đồ thị có hướng  $G$  ( $n, m \leq 10^6$ )
- ✦  $m$  dòng tiếp theo, mỗi dòng chứa hai số nguyên  $u, v$  cho biết có cung  $(u, v)$  trên đồ thị

### Output

- ✦ Danh sách các đỉnh có thể đến được từ  $s$
- ✦ Đường đi từ  $s$  tới  $t$  nếu có

| Sample Input | Sample Output              |
|--------------|----------------------------|
| 10 12 0 7    | Reachable vertices from 0: |
| 0 1          | 0, 1, 2, 4, 6, 7, 3, 5,    |
| 0 2          | The path from 0 to 7:      |
| 1 2          | 7 <- 6 <- 2 <- 1 <- 0      |
| 1 3          |                            |
| 2 0          |                            |
| 2 4          |                            |
| 2 6          |                            |
| 3 5          |                            |
| 3 6          |                            |
| 5 1          |                            |
| 6 7          |                            |
| 8 9          |                            |



## 1.2. Biểu diễn đồ thị

Mặc dù đồ thị được cho trong input dưới dạng danh sách cạnh, đối với những thuật toán trong bài, cách biểu diễn đồ thị hiệu quả nhất là sử dụng danh sách kề hoặc danh sách liên thuộc: Mỗi đỉnh  $u$  tương ứng với một danh sách  $adj[u]$  chứa các đỉnh nối từ  $u$  (danh sách kề dạng forward star).

## 1.3. Thuật toán tìm kiếm theo chiều sâu

### 1.3.1. Ý tưởng

Tư tưởng của *thuật toán tìm kiếm theo chiều sâu* (Depth-First Search – DFS) có thể trình bày như sau: Trước hết, dĩ nhiên đỉnh  $s$  đến được từ  $s$ , tiếp theo, với mọi cung  $(s, x)$  của đồ thị thì  $x$  cũng sẽ đến được từ  $s$ . Với mỗi đỉnh  $x$  đó thì tất nhiên những đỉnh  $y$  nối từ  $x$  cũng đến được từ  $s$ ... Điều đó gợi ý cho ta viết một hàm đệ quy DFSVisit( $u$ ) mô tả việc duyệt từ đỉnh  $u$  bằng cách thăm đỉnh  $u$  và tiếp tục quá trình duyệt DFSVisit( $v$ ) với  $v$  là một đỉnh chưa thăm nối từ  $u$ .

Kỹ thuật đánh dấu được sử dụng để tránh việc liệt kê lặp các đỉnh: Khởi tạo  $avail[v] = \text{true}, \forall v \in V$ , mỗi lần thăm một đỉnh, ta đánh dấu đỉnh đó lại ( $avail[v] = \text{false}$ ) để các bước duyệt đệ quy kế tiếp không duyệt lại đỉnh đó nữa.

Để lưu lại đường đi từ đỉnh xuất phát  $s$ , trong hàm DFSVisit( $u$ ), trước khi gọi đệ quy DFSVisit( $v$ ) với  $v$  là một đỉnh chưa thăm nối từ  $u$  (chưa đánh dấu), ta lưu lại vết đường đi từ  $u$  tới  $v$  bằng cách đặt  $trace[v] = u$ , tức là  $trace[v]$  lưu lại đỉnh liền trước  $v$  trong đường đi từ  $s$  tới  $v$ . Khi thuật toán DFS kết thúc, đường đi từ  $s$  tới  $t$  sẽ là:


$$\langle p_0 = t \leftarrow p_1 = trace[p_0] \leftarrow p_2 = trace[p_1] \leftarrow \dots \leftarrow s \rangle$$

```

1 void DFSVisit(u ∈ V)
2 {
3     avail[u] = false; //Đánh dấu avail[u] = false ⇔ u đã thăm
4     Output ← u; //liệt kê u
5     for (∀v ∈ adj[u]) //duyệt mọi đỉnh v nối từ u
6         if (avail[v]) //nếu v chưa thăm
7             {
8                 trace[v] = u; //Lưu vết: u đứng liền trước v trên đường đi s ~ v
9                 DFSVisit(v); //Gọi đệ quy tìm kiếm theo chiều sâu từ v
10            }
11 }
12
13 Input → đồ thị G, đỉnh xuất phát s, đỉnh cần đến t;
14 for (∀u ∈ V) //Đánh dấu các đỉnh đều chưa thăm
15     avail[u] = true;
16 DFSVisit(s);
17 if (!avail[t]) //Từ s có đường tới t
18     «Truy theo vết từ t để tìm đường đi từ s tới t»;

```

### 1.3.2. Cài đặt

DFS.cpp  Tìm đường bằng DFS

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 const int maxN = 1e6;
5
6 int n, m, s, t, trace[maxN];
7 bool avail[maxN];
8 vector<int> adj[maxN];
9
10 void ReadInput() //Nhập dữ liệu
11 {
12     cin >> n >> m >> s >> t;
13     while (m-- > 0)
14     {
15         int u, v;
16         cin >> u >> v; //Đọc một cạnh (u, v)
17         adj[u].push_back(v); //Đưa v vào danh sách kề của u
18     }
19 }
20
21 void DFSVisit(int u) //Thuật toán tìm kiếm theo chiều sâu bắt đầu từ u
22 {
23     avail[u] = false; //Đánh dấu u đã thăm
24     cout << u << ", "; //Liệt kê u
25     for (int v: adj[u]) //Xét mọi đỉnh v nối từ u
26         if (avail[v]) //Nếu v chưa thăm
27             {
28                 trace[v] = u; //Lưu vết đường đi s → ... → u → v
29                 DFSVisit(v); //Đệ quy tìm kiếm theo chiều sâu bắt đầu từ v
30             }
31 }
32

```

```

33 void PrintPath() //In đường đi
34 {
35     if (avail[t]) //t chưa thăm: không có đường từ s tới t
36         cout << "There's no path from " << s << " to " << t << '\n';
37     else
38     {
39         cout << "The path from " << s << " to " << t << ":\n";
40         for (int u = t; u != s; u = trace[u]) //Truy vết ngược từ t về s
41             cout << u << " <- ";
42         cout << s << "\n";
43     }
44 }
45
46 int main()
47 {
48     ReadInput();
49     fill(avail, avail + n, true); //Khởi tạo các đỉnh đều chưa thăm
50     cout << "Reachable vertices from " << s << ":\n";
51     DFSVisit(s); //Thuật toán tìm kiếm theo chiều sâu bắt đầu từ s
52     cout << "\n";
53     PrintPath();
54 }

```

Về kỹ thuật, có thể không cần mảng đánh dấu *avail*[0 ... *n*) mà dùng luôn mảng *trace*[0 ... *n*) để đánh dấu: Khởi tạo các phần tử mảng *trace*[0 ... *n*) là:

$$\begin{cases} trace[s] \neq -1 \\ trace[v] = -1, \forall v \neq s \end{cases}$$

Khi đó điều kiện để một đỉnh *v* chưa thăm là *trace*[*v*] = -1, mỗi khi từ đỉnh *u* thăm đỉnh *v*, phép gán *trace*[*v*] = *u* sẽ kiêm luôn công việc đánh dấu *v* đã thăm (*trace*[*v*] ≠ -1).

Chương trình cài đặt in ra các đỉnh đường đi theo thứ tự ngược (từ *t* về *s*). Để in đường đi theo đúng thứ tự từ *s* tới *t*, có thể sử dụng một số kỹ thuật đơn giản:

Cách 1: Lưu đường đi vào mảng và in mảng ra theo thứ tự ngược lại.

Cách 2: Viết một hàm đệ quy DoTrace(*t*) để truy vết:

```

1 void DoTrace(int v) //In ra đường đi từ s tới v
2 {
3     if (v != s)
4     {
5         DoTrace(trace[v]); //In ra đường đi từ s tới trace[v] trước
6         cout << " -> ";
7     }
8     cout << v; //Sau đó in ra v
9 }

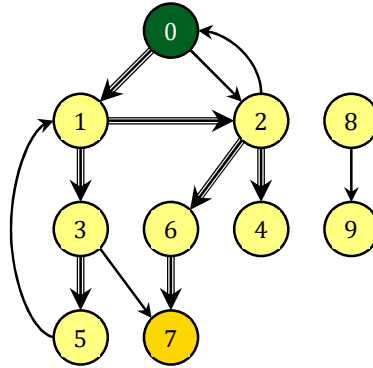
```

Cách 3: Thay danh sách kề dạng forward star bằng danh sách kề dạng reverse star rồi thực hiện thuật toán tìm kiếm theo chiều sâu bắt đầu từ *t*, sau đó truy vết từ *s*.

### 1.3.3. Một vài tính chất của DFS

#### \* Cây DFS

Nếu ta sắp xếp danh sách kề của mỗi đỉnh theo thứ tự tăng dần thì thuật toán DFS luôn trả về đường đi có thứ tự từ điển nhỏ nhất trong số tất cả các đường đi từ  $s$  tới tới  $t$ .



Hình 1-2. Cây DFS

Quá trình tìm kiếm theo chiều sâu cho ta một cây DFS gốc  $s$ . Quan hệ cha-con trên cây được định nghĩa là: nếu từ đỉnh  $u$  tới thăm đỉnh  $v$  ( $\text{DFSVisit}(u)$  gọi  $\text{DFSVisit}(v)$ ) thì  $u$  là nút cha của nút  $v$ . Hình 1-2 là đồ thị với danh sách kề của mỗi đỉnh được sắp xếp tăng dần và cây DFS tương ứng với đỉnh xuất phát  $s = 0$ . Chú ý rằng cấu trúc của cây DFS phụ thuộc vào thứ tự trong các danh sách kề.

#### \* Mô hình duyệt đồ thị theo DFS

Trong bài toán này, thực ra ta mới chỉ khảo sát một ứng dụng của thuật toán DFS để liệt kê các đỉnh đến được từ một đỉnh  $s$  cho trước. Mô hình cài đặt đầy đủ của thuật toán DFS cho phép duyệt qua tất cả các đỉnh cũng như tất cả các cạnh của đồ thị:

```
1 void DFSVisit( $u \in V$ )
2 {
3      $d[u] = ++\text{Time}$ ; //  $d[u]$  = thời điểm  $u$  được thăm, cũng là đánh dấu  $\neq 0$ 
4      $\text{Output} \leftarrow u$ ; // Thăm tới đỉnh nào in ra luôn đỉnh đó
5     for ( $\forall v \in \text{adj}[u]$ ) // duyệt mọi đỉnh  $v$  nối từ  $u$ 
6         if ( $d[v] == 0$ ) // nếu  $v$  chưa thăm
7             DFSVisit( $v$ ); // Gọi đệ quy tìm kiếm theo chiều sâu từ  $v$ 
8      $f[u] = ++\text{Time}$ ; //  $f[u]$  = thời điểm  $u$  được duyệt xong
9 }
10
11  $\text{Input} \rightarrow$  đồ thị  $G$ ;
12 for ( $\forall u \in V$ ) // Khởi tạo các đỉnh đều chưa thăm:  $d[.] = 0$ 
13      $d[u] = 0$ ;
14  $\text{Time} = 0$ ;
15 for ( $\forall u \in V$ ) // Xét mọi đỉnh chưa thăm
16     if ( $d[u] == 0$ )
17         DFSVisit( $u$ );
```

### ✧ Thời gian thực hiện giải thuật

Với mỗi đỉnh  $u$  thì hàm  $DFSVisit(u)$  được gọi đúng 1 lần và trong hàm đó ta có chi phí xét tất cả các đỉnh  $v$  nối từ  $u$ , cũng là chi phí quét tất cả các cung đi ra khỏi  $u$ .

- ✧ Nếu đồ thị được biểu diễn bằng danh sách kề hoặc danh sách liên thuộc, mỗi cung của đồ thị sẽ bị duyệt qua đúng 1 lần, trong trường hợp này, thời gian thực hiện giải thuật DFS là  $\Theta(n + m)$  bao gồm  $n$  lần gọi hàm  $DFSVisit(.)$  và  $m$  lần duyệt qua cung.
- ✧ Nếu đồ thị được biểu diễn bằng ma trận kề, để xét tất cả các đỉnh nối từ một đỉnh  $u$  ta phải quét qua toàn bộ tập đỉnh mất thời gian  $\Theta(n)$ . Trong trường hợp này thời gian thực hiện giải thuật DFS là  $\Theta(n^2)$ .
- ✧ Nếu đồ thị được biểu diễn bằng danh sách cạnh, để xét tất cả các đỉnh nối từ một đỉnh  $u$  ta phải quét toàn bộ danh sách cạnh mất thời gian  $\Theta(m)$ . Trong trường hợp này thời gian thực hiện giải thuật DFS là  $\Theta(n \cdot m)$ .

### ✧ Thứ tự thăm đến và duyệt xong

Trong mô hình duyệt đồ thị bằng DFS, hãy để ý hàm  $DFSVisit(u)$ :

- ✧ Khi bắt đầu vào hàm ta nói đỉnh  $u$  được *thăm đến* (*discover*), có nghĩa là tại thời điểm đó, quá trình tìm kiếm theo chiều sâu bắt đầu từ  $u$  sẽ xây dựng nhánh cây DFS gốc  $u$ .
- ✧ Khi chuẩn bị thoát khỏi hàm để lùi về , ta nói đỉnh  $u$  được *duyet xong* (*finish*), có nghĩa là tại thời điểm đó, quá trình tìm kiếm theo chiều sâu từ  $u$  kết thúc.

Trong mô hình duyệt đồ thị bằng DFS, một biến đếm  $Time$  được dùng để xác định thời điểm thăm đến  $d[u]$  và thời điểm duyệt xong  $f[u]$  của mỗi đỉnh  $u$ . Thứ tự thăm đến và duyệt xong này có ý nghĩa rất quan trọng trong nhiều thuật toán có áp dụng DFS, chẳng hạn như các thuật toán tìm thành phần liên thông mạnh, thuật toán sắp xếp tô pô...

### **Bổ đề 1-1**

Với hai đỉnh phân biệt  $u, v$ :

- ✧ Đỉnh  $v$  được thăm đến trong thời gian từ  $d_u$  đến  $f_u$ :  $d_v \in [d_u, f_u]$  nếu và chỉ nếu  $v$  là hậu duệ của  $u$  trên cây DFS.
- ✧ Đỉnh  $v$  được duyệt xong trong thời gian từ  $d_u$  đến  $f_u$ :  $f_v \in [d_u, f_u]$  nếu và chỉ nếu  $v$  là hậu duệ của  $u$  trên cây DFS.

### **Chứng minh**

Bản chất của việc đỉnh  $v$  được thăm đến (hay duyệt xong) trong thời gian từ  $d_u$  đến  $f_u$  chính là hàm  $DFSVisit(v)$  được gọi (hay thoát) khi mà hàm  $DFSVisit(u)$  đã bắt đầu nhưng chưa kết thúc, nghĩa là hàm  $DFSVisit(v)$  được dây chuyền đệ quy từ  $DFSVisit(u)$  gọi tới. Điều này chỉ ra rằng  $v$  nằm trong nhánh DFS gốc  $u$ , hay nói cách khác,  $v$  là hậu duệ của  $u$ .

### **Hệ quả**

Với hai đỉnh phân biệt  $(u, v)$  thì hai đoạn  $[d_u, f_u]$  và  $[d_v, f_v]$  hoặc rời nhau hoặc chứa nhau. Hai đoạn  $[d_u, f_u]$  và  $[d_v, f_v]$  chứa nhau nếu và chỉ nếu  $u$  và  $v$  có quan hệ tiền bối-hậu duệ.

### **Chứng minh**

Dễ thấy rằng nếu hai đoạn  $[d_u, f_u]$  và  $[d_v, f_v]$  không rời nhau thì hoặc  $d_u \in [d_v, f_v]$  hoặc  $d_v \in [d_u, f_u]$ , tức là hai đỉnh  $(u, v)$  có quan hệ tiền bối-hậu duệ, áp dụng Bổ đề 1-1, ta có ĐPCM.

### **Bổ đề 1-2**

Với hai đỉnh phân biệt  $u \neq v$  mà  $(u, v) \in E$  thì  $v$  phải được thăm đến trước khi  $u$  được duyệt xong:

$$(u, v) \in E \Rightarrow d_v < f_u$$

### **Chứng minh**

Đây là một tính chất quan trọng của thuật toán DFS. Hãy để ý hàm  $DFSVisit(u)$ , trước khi thoát (duyet xong  $u$ ), nó sẽ quét tất cả các đỉnh chưa thăm nối từ  $u$  và gọi đệ quy để thăm những đỉnh đó, tức là nếu  $(u, v) \in E$ ,  $v$  phải được thăm đến trước khi  $u$  được duyệt xong:  $d_v < f_u$ .

### **Định lý 1-3 (định lý đường đi trắng)**

Đỉnh  $v$  là hậu duệ thực sự của đỉnh  $u$  trong một cây DFS nếu và chỉ nếu tại thời điểm  $d_u$  mà thuật toán thăm tới đỉnh  $u$ , tồn tại một đường đi từ  $u$  tới  $v$  mà ngoại trừ đỉnh  $u$ , tất cả các đỉnh khác trên đường đi đều chưa được thăm.

### **Chứng minh**

“ $\Rightarrow$ ”

Nếu  $v$  là hậu duệ của  $u$ , ta xét đường đi từ  $u$  tới  $v$  dọc trên các cung trên cây DFS. Tất cả các đỉnh  $w$  nằm sau  $u$  trên đường đi này đều là hậu duệ của  $u$ , nên theo Bổ đề 1-1, ta có  $d_u < d_w$ , tức là vào thời điểm  $d_u$ , tất cả các đỉnh  $w$  đó đều chưa được thăm

“ $\Leftarrow$ ”

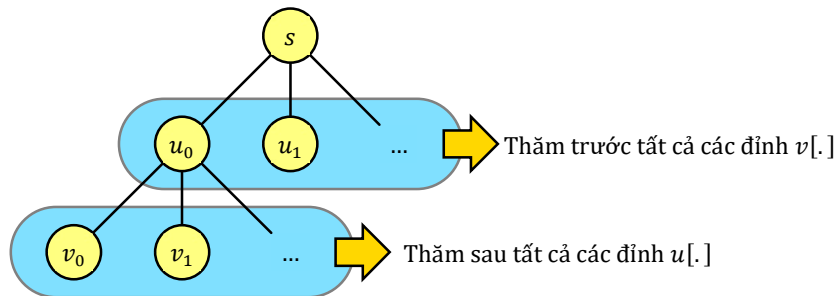
Nếu tại thời điểm  $d_u$ , tồn tại một đường đi từ  $u$  tới  $v$  mà tất cả các đỉnh khác  $u$  trên đường đi đều chưa được thăm, ta sẽ chứng minh rằng mọi đỉnh trên đường đi này đều là hậu duệ của  $u$ . Thật vậy, giả sử phản chứng rằng  $y$  là đỉnh đầu tiên trên đường đi này mà không phải hậu duệ của  $u$ , tức là tồn tại đỉnh  $x$  liền trước  $y$  trên đường đi là hậu duệ của  $u$ . Theo Bổ đề 1-2,  $y$  phải được thăm trước khi duyệt xong  $x$ :  $d_y < f_x$ ;  $x$  lại là hậu duệ của  $u$  nên theo Bổ đề 1-1, ta có  $f_x \leq f_u$ , vậy  $d_y < f_u$ . Mặt khác theo giả thiết rằng tại thời điểm  $d_u$  thì  $y$  chưa được thăm, tức là  $d_u < d_y$ , kết hợp lại ta có  $d_u < d_y < f_u$ , vậy thì  $y$  là hậu duệ của  $u$  theo Bổ đề 1-1, trái với giả thiết phản chứng.

Tên gọi “định lý đường đi trắng: white-path theorem” xuất phát từ cách trình bày thuật toán DFS bằng cơ chế tô màu đồ thị: Ban đầu các đỉnh được tô màu trắng, mỗi khi thăm đến một đỉnh thì đỉnh đó được tô màu xám và mỗi khi duyệt xong một đỉnh thì đỉnh đó được tô màu đen: Định lý khi đó có thể phát biểu: Điều kiện cần và đủ để đỉnh  $v$  là hậu duệ thực sự của đỉnh  $u$  trong một cây DFS là tại thời điểm đỉnh  $u$  được tô màu xám, tồn tại một đường đi từ  $u$  tới  $v$  mà ngoại trừ đỉnh  $u$ , tất cả các đỉnh khác trên đường đi đều có màu trắng.

## 1.4. Thuật toán tìm kiếm theo chiều rộng

### 1.4.1. Ý tưởng

Thực ra ta đã biết về thuật toán này trong khi tìm hiểu ứng dụng của cấu trúc dữ liệu hàng đợi (thuật toán loang). Tư tưởng của thuật toán tìm kiếm theo chiều rộng (Breadth-First Search – BFS) là “lập lịch” duyệt các đỉnh. Việc thăm một đỉnh sẽ lên lịch duyệt các đỉnh nối từ nó sao cho thứ tự duyệt là ưu tiên chiều rộng (đỉnh nào gần đỉnh xuất phát  $s$  hơn sẽ được duyệt trước). Đầu tiên ta thăm đỉnh  $s$ . Việc thăm đỉnh  $s$  sẽ phát sinh thứ tự thăm những đỉnh  $u_0, u_1, \dots$  nối từ  $s$  (những đỉnh gần  $s$  nhất). Tiếp theo ta thăm đỉnh  $u_0$ , khi thăm đỉnh  $u_0$  sẽ lại phát sinh yêu cầu thăm những đỉnh  $v_0, v_1, \dots$  nối từ  $u_0$ . Nhưng rõ ràng các đỉnh  $v[.]$  này “xa”  $s$  hơn những đỉnh  $u[.]$  nên chúng chỉ được thăm khi tất cả những đỉnh  $u[.]$  đã thăm. Tức là thứ tự duyệt đỉnh sẽ là:  $s, u_0, u_1, \dots, v_0, v_1, \dots$  (Hình 1-3).



Hình 1-3. Thứ tự thăm đỉnh của BFS

Thuật toán tìm kiếm theo chiều rộng sử dụng một danh sách để chứa những đỉnh đang “chờ” thăm. Tại mỗi bước, ta thăm một đỉnh đầu danh sách, loại nó ra khỏi danh sách và cho những đỉnh chưa “xếp hàng” kề với nó xếp hàng thêm vào cuối danh sách. Thuật toán sẽ kết thúc khi danh sách rỗng. Vì nguyên tắc vào trước ra trước, danh sách chứa những đỉnh đang chờ thăm được tổ chức dưới dạng hàng đợi (queue).




```

1 Input → đồ thị G, đỉnh xuất phát s, đỉnh cần đến t;
2 Queue = {s};
3 for (∀u ∈ V)
4     avail[u] = true; //Các đỉnh đều đánh dấu chưa xếp hàng
5 avail[s] = false; //Riêng đỉnh s được xếp hàng
6 do //Lặp chừng nào hàng đợi khác rỗng
7 {
8     u = Queue.front(); Queue.pop(); //Lấy u khỏi hàng đợi
9     for (∀v ∈ adj[u]) //Duyệt các đỉnh v nối từ u
10         if (avail[v]) //Nếu v chưa xếp hàng
11         {
12             Queue.push(v); //Cho v xếp hàng trong hàng đợi
13             avail[v] = false;
14             trace[v] = u; //Lưu vết: u đứng liền trước v trên đường đi s ~ v
15         }
16 }
17 while (Queue ≠ ∅);
18 if (!avail[t]) //Từ s có đường tới t
19     «Truy theo vết từ t để tìm đường đi từ s tới t»;

```

### 1.4.2. Cài đặt

BFS.cpp  Tìm đường bằng BFS

```

1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 using namespace std;
5 const int maxN = 1e6;
6
7 int n, m, s, t, trace[maxN];
8 bool avail[maxN];
9 vector<int> adj[maxN];
10
11 void ReadInput() //Nhập dữ liệu
12 {
13     cin >> n >> m >> s >> t;
14     while (m-- > 0)
15     {
16         int u, v;
17         cin >> u >> v; //Đọc một cạnh (u, v)
18         adj[u].push_back (v); //Đưa v vào danh sách kề của u
19     }
20 }
21

```

```

22 void BFS() //Thuật toán tìm kiếm theo chiều rộng
23 {
24     queue<int> Queue({s}); //Hàng đợi ban đầu chỉ gồm một đỉnh s
25     fill (avail, avail + n, true); //Khởi tạo các đỉnh đều chưa thăm
26     avail[s] = false; //Riêng đỉnh s đã thăm
27     do
28     {
29         int u = Queue.front(); Queue.pop(); //Lấy u từ hàng đợi
30         cout << u << ", ";
31         for (int v: adj[u]) //Xét mọi đỉnh v nối từ u
32             if (avail[v]) //Nếu v chưa thăm
33             {
34                 trace[v] = u; //Lưu vết đường đi s → ... → u → v
35                 Queue.push(v); //Đẩy v vào hàng đợi
36                 avail[v] = false;
37             }
38     }
39     while (!Queue.empty());
40 }
41
42 void PrintPath() //In đường đi
43 {
44     if (avail[t]) //t chưa thăm: không có đường từ s tới t
45         cout << "There's no path from " << s << " to " << t << '\n';
46     else
47     {
48         cout << "The path from " << s << " to " << t << ":\n";
49         for (int u = t; u != s; u = trace[u]) //Truy vết ngược từ t về s
50             cout << u << " <- ";
51         cout << s << "\n";
52     }
53 }
54
55 int main()
56 {
57     ReadInput();
58     cout << "Reachable vertices from " << s << ":\n";
59     BFS(); //Thuật toán tìm kiếm theo chiều rộng
60     cout << "\n";
61     PrintPath();
62 }

```

Tương tự như thuật toán tìm kiếm theo chiều sâu, ta có thể dùng mảng  $trace[0 \dots n)$  kiểm tra luôn chức năng đánh dấu.

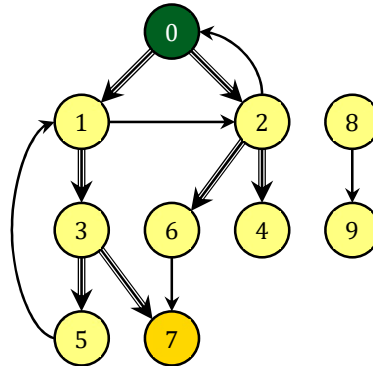
### 1.4.3. Một vài tính chất của BFS

#### ✱ Cây BFS

Thuật toán BFS luôn trả về đường đi qua ít cạnh nhất trong số tất cả các đường đi từ  $s$  tới  $t$ . Nếu các danh sách kề được sắp xếp theo thứ tự tăng dần thì thuật toán BFS sẽ trả về đường đi có thứ tự từ điển nhỏ nhất trong số những đường đi qua ít cạnh nhất.

Quá trình tìm kiếm theo chiều rộng cho ta một cây BFS gốc  $s$ . Quan hệ cha-con trên cây được định nghĩa là: nếu từ đỉnh  $u$  tới thăm đỉnh  $v$  (khi đỉnh  $u$  được rút ra khỏi

hàng đợi thì  $v$  là một trong số những đỉnh chưa thăm nối từ  $u$  được xét đến và đẩy vào hàng đợi) thì  $u$  là nút cha của nút  $v$ .



Hình 1-4. Cây BFS

Hình 1-4 là đồ thị với danh sách kề của mỗi đỉnh được sắp xếp tăng dần và cây BFS tương ứng với đỉnh xuất phát  $s = 0$ . Chú ý rằng cấu trúc của cây BFS cũng phụ thuộc vào thứ tự trong các danh sách kề.

#### \* Mô hình duyệt đồ thị theo BFS

Tương tự như thuật toán DFS, mô hình tổng quát của thuật toán BFS cũng dùng để xác định một thứ tự trên các đỉnh của đồ thị:

```

1 void BFSVisit( $s \in V$ )
2 {
3     Queue = ( $s$ ); //hàng đợi chỉ gồm một đỉnh  $s$ 
4      $d[s] = ++Time$ ; // $d[s] = 1$ , cũng là đánh dấu  $\neq 0$ 
5     do
6     {
7          $u = \text{Queue.front}()$ ;  $\text{Queue.pop}()$ ; //Lấy  $u$  khỏi hàng đợi
8          $f[u] = ++Time$ ; //Ghi nhận thời điểm duyệt xong  $u$ 
9         Output  $\leftarrow u$ ; //Duyệt xong đỉnh nào in ra luôn đỉnh đó
10        for ( $\forall v \in \text{adj}[u]$ ) //duyet mọi đỉnh  $v$  nối từ  $u$ 
11            if ( $d[v] == 0$ ) //nếu  $v$  chưa thăm
12            {
13                Queue.push( $v$ );
14                 $d[v] = ++Time$ ; //thời điểm thăm đến  $v$ , cũng là đánh dấu  $d[v] \neq 0$ 
15            }
16        }
17    while (Queue  $\neq \emptyset$ );
18 }
19
20 Input  $\rightarrow$  đồ thị  $G$ ;
21 for ( $\forall u \in V$ ) //Các đỉnh đều chưa thăm:  $d[.] = 0$ 
22      $d[u] = 0$ ;
23 Time = 0;
24 for ( $\forall u \in V$ ) //Xét mọi điểm chưa thăm
25     if ( $d[u] == 0$ )
26         BFSVisit( $u$ );

```

### ✳ Thời gian thực hiện giải thuật

Thời gian thực hiện giải thuật của BFS tương tự như đối với DFS, bằng  $\Theta(|V| + |E|)$  nếu đồ thị được biểu diễn bằng danh sách kề hoặc danh sách liên thuộc, bằng  $\Theta(|V|^2)$  nếu đồ thị được biểu diễn bằng ma trận kề, và bằng  $\Theta(|V||E|)$  nếu đồ thị được biểu diễn bằng danh sách cạnh.

### ✳ Thứ tự thăm đến và duyệt xong

Trong mô hình duyệt đồ thị bằng BFS, người ta cũng quan tâm tới thứ tự thăm đến và duyệt xong: Khi một đỉnh được đẩy vào hàng đợi, ta nói đỉnh đó được thăm đến (được thăm) và khi một đỉnh được lấy ra khỏi hàng đợi, ta nói đỉnh đó được duyệt xong. Cụ thể là, mỗi đỉnh  $u$  sẽ tương ứng với thời điểm thăm đến  $d_u$  và thời điểm duyệt xong  $f_u$ .

Bây giờ ta thử sửa đổi mô hình cài đặt: thay cơ chế đánh dấu thăm đến/chưa thăm đến bằng duyệt xong/chưa duyệt xong:

```
1 void BFSVisit(s ∈ V)
2 {
3     Queue = (s); //Hàng đợi chỉ gồm một đỉnh s
4     do
5     {
6         u = Queue.front(); Queue.pop(); //Lấy u khỏi hàng đợi
7         if (f[u] != 0) //u đã duyệt xong, bỏ qua
8             f[u] = ++Time; //Ghi nhận thời điểm duyệt xong u, cũng là đánh dấu ≠ 0
9         Output ← u; //Duyệt xong đỉnh nào in ra luôn đỉnh đó
10        for (∀v ∈ adj[u]) //duyet mọi đỉnh v nối từ u
11            if (f[v] == 0) //nếu v chưa duyệt xong
12                Queue.push(v);
13    }
14    while (Queue ≠ ∅);
15 }
16
17 Input → đồ thị G;
18 for (∀u ∈ V) //Các đỉnh đều chưa duyệt xong: f[.] = 0
19     f[u] = 0;
20 Time = 0;
21 for (∀u ∈ V) //Xét mọi điểm chưa duyệt xong
22     if (f[u] == 0)
23         BFSVisit(u);
```

Thứ tự các đỉnh được in ra không thay đổi, tuy nhiên có sự thay đổi trong quy trình xử lý:

- ✳ Ở cách cài đặt dùng cơ chế đánh dấu thăm/chưa thăm, mỗi đỉnh được đẩy vào hàng đợi đúng một lần và bị lấy ra đúng một lần
- ✳ Ở cách cài đặt dùng cơ chế duyệt xong/chưa duyệt xong, một đỉnh có thể bị đẩy vào cũng như lấy ra khỏi hàng đợi nhiều lần (tối đa  $\deg^-(v)$  lần đối với đỉnh  $v$ ), hàng đợi có thể có thời điểm phải chứa nhiều hơn  $n$  đỉnh (tuy nhiên kích thước hàng đợi không bao giờ vượt quá  $m$ ).

Điều đặc biệt là trong mô hình cài đặt dùng cơ chế đánh dấu duyệt xong/chưa duyệt xong, nếu ta:

- ✿ Thay cấu trúc hàng đợi bởi cấu trúc ngăn xếp
- ✿ Lật ngược thứ tự đỉnh trong các danh sách kề

khi đó thuật toán sẽ in ra các đỉnh theo thứ tự duyệt đỉnh DFS. Đây chính là phương pháp khử đệ quy của DFS để cài đặt thuật toán trên các ngôn ngữ không cho phép đệ quy.

---

### Bài tập 1-1

Viết chương trình cài đặt thuật toán DFS không đệ quy.

### Bài tập 1-2

Xét đồ thị có hướng  $G = (V, E)$ , dùng thuật toán DFS duyệt đồ thị  $G$ . Cho một phản ví dụ để chứng minh giả thuyết sau là sai: Nếu từ đỉnh  $u$  có đường đi tới đỉnh  $v$  và  $u$  được thăm đến trước  $v$ , thì  $v$  nằm trong nhánh DFS gốc  $u$ .

### Bài tập 1-3

Cho đồ thị vô hướng  $G = (V, E)$ , tìm thuật toán  $O(|V|)$  để phát hiện một chu trình đơn trong  $G$ .

### Bài tập 1-4

Cho đồ thị có hướng  $G = (V, E)$  có  $n$  đỉnh, và mỗi đỉnh  $i$  được gán một nhãn là số nguyên  $a_i$ , tập cung  $E$  của đồ thị được định nghĩa là  $(u, v) \in E \Leftrightarrow a_u \geq a_v$ . Giả sử rằng thuật toán DFS được sử dụng để duyệt đồ thị, hãy khảo sát tính chất của dãy các nhãn nếu ta xếp các đỉnh theo thứ tự từ đỉnh duyệt xong đầu tiên đến đỉnh duyệt xong sau cùng.

### Bài tập 1-5

Mê cung hình chữ nhật kích thước  $m \times n$  gồm các ô vuông đơn vị. Trên mỗi ô ghi một trong ba ký tự:

- ✿ O: Nếu ô đó an toàn
- ✿ X: Nếu ô đó có cạm bẫy
- ✿ E: Nếu là ô có một nhà thám hiểm đang đứng.

Duy nhất chỉ có 1 ô ghi chữ E. Nhà thám hiểm có thể từ một ô đi sang một trong số các ô chung cạnh với ô đang đứng. Một cách đi thoát khỏi mê cung là một hành trình đi qua các ô an toàn ra một ô biên. Hãy chỉ giúp cho nhà thám hiểm một hành trình thoát ra khỏi mê cung đi qua ít ô nhất. Yêu cầu thuật toán có độ phức tạp  $O(mn)$ .