Toán rời rạc 2

Discrete mathematics 2

Bài 3: Tìm kiếm trên đồ thị

Graph Traversal

Link(s) download slide bài giảng

0. Giới thiệu (https://bit.ly/3byPOri)

   Introduction

1. Các khái niệm cơ bản (https://bit.ly/2UgGPFN)

   Terminology

2. Biểu diễn đồ thị (https://bit.ly/31g0ffy)

   Representing Graphs

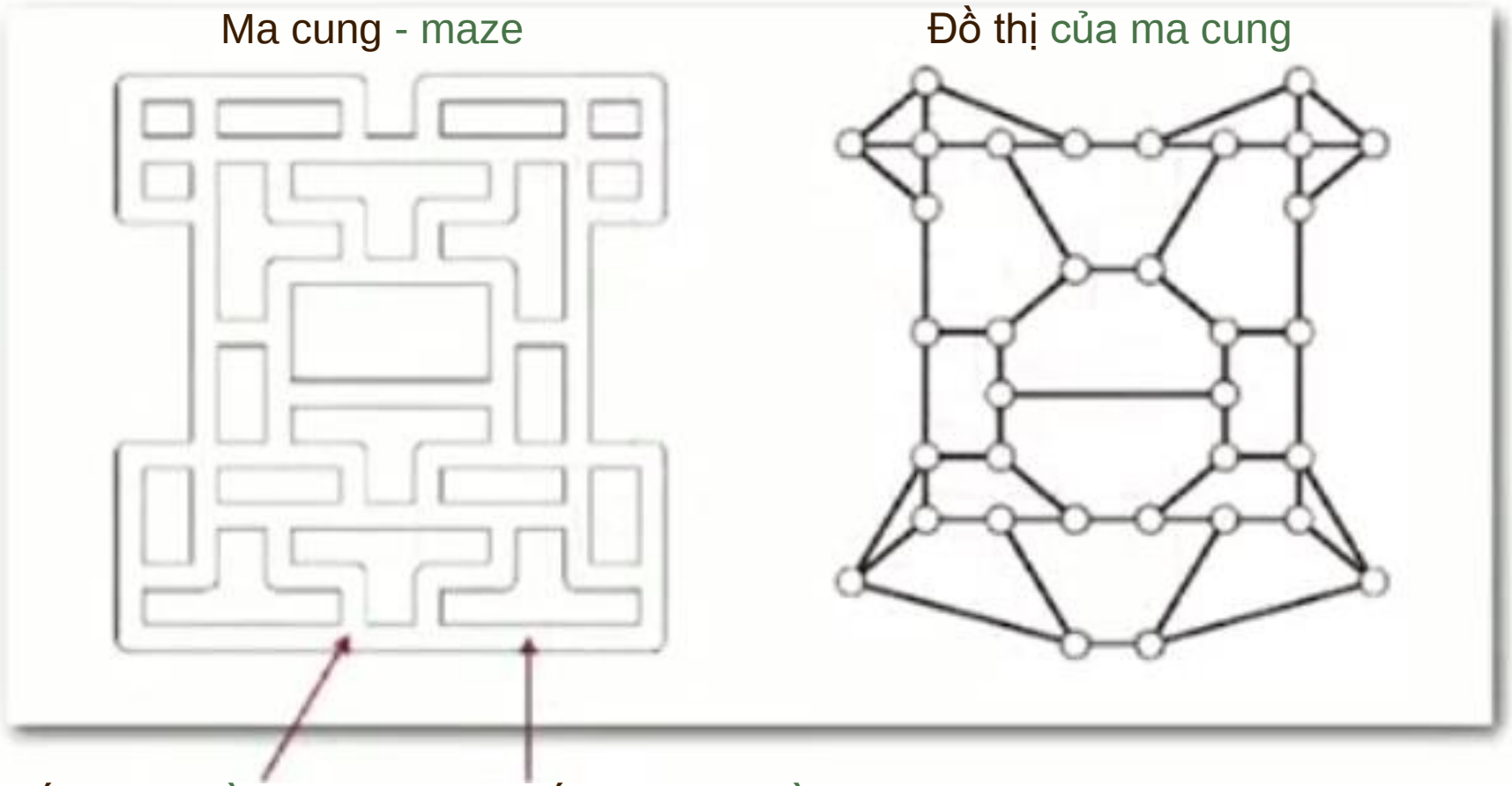
3. Tìm kiếm - duyệt đồ thị (https://bit.ly/3cq8TNQ)

   Graph Traversal

... (tiếp tục cập nhật)

Tìm kiếm - duyệt đồ thị (1/2)

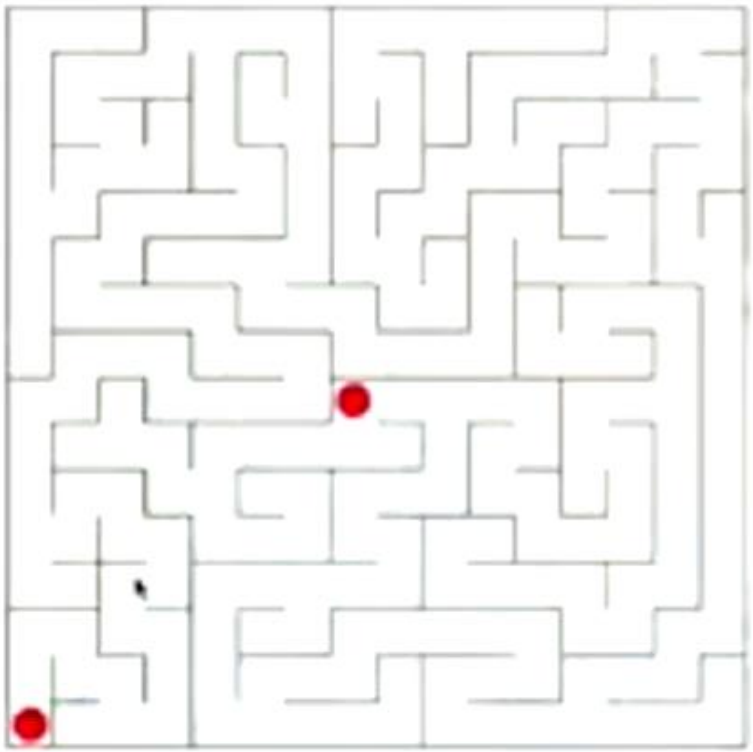
- Thám hiểm tất cả các giao cắt của ma cung

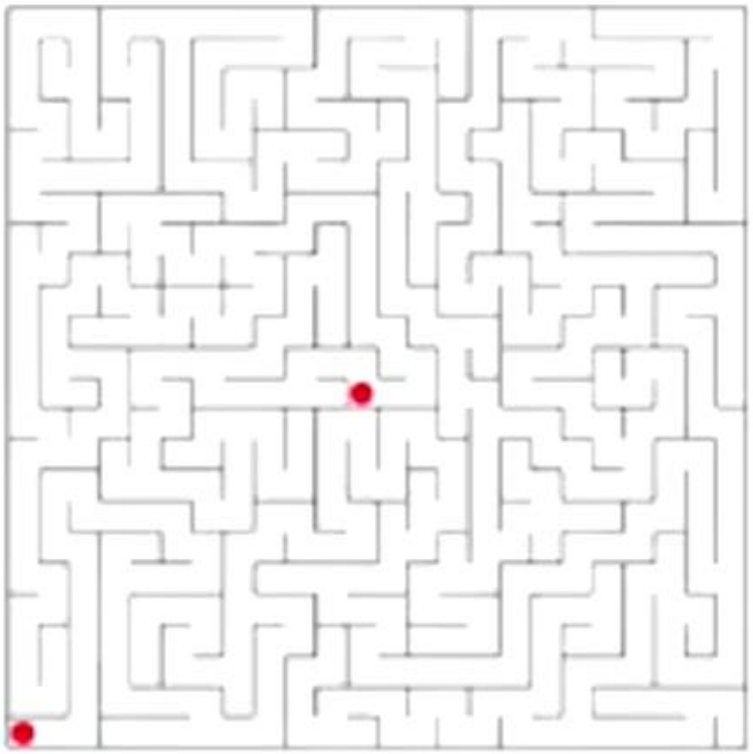


Tìm kiếm - duyệt đồ thị (2/2)

Khám phá ma cung ⇨ tìm đường đi.

⇔ Duyệt các điểm giao cắt





Nội dung Bài 3

1. Thuật toán tìm kiếm theo chiều sâu
   Depth-First Search - DFS

2. Thuật toán tìm kiếm theo chiều rộng
   Breadth-First Search - BFS

3. Một số ứng dụng của DFS và BFS

Tìm kiếm theo chiều sâu - DFS

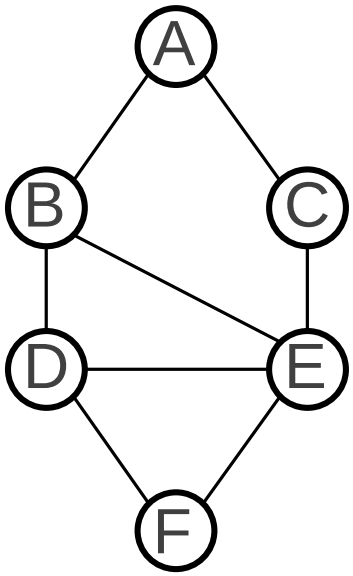
- Tư tưởng
  - Quá trình tìm kiếm ưu tiên "chiều sâu" hơn "chiều rộng"
  - Đi xuống sâu nhất có thể trước khi quay lại
- Thuật toán đệ quy

```
DFS(u){                          //u là đỉnh bắt đầu duyệt
    <Thăm đỉnh u>;               //duyệt đỉnh u
    chuaxet[u] = false;          //xác nhận đỉnh u đã duyệt
    for(v ∈ Ke(u)){
        if(chuaxet[v])           //nếu v chưa được duyệt
            DFS(v);              //duyệt theo chiều sâu từ v
    }
}
```

chuaxet[A] = T	chuaxet[A] = F
chuaxet[B] = T	chuaxet[B] = F
chuaxet[C] = T	chuaxet[C] = F
chuaxet[D] = T	chuaxet[D] = F
chuaxet[E] = T	chuaxet[E] = F
chuaxet[F] = T	chuaxet[F] = F

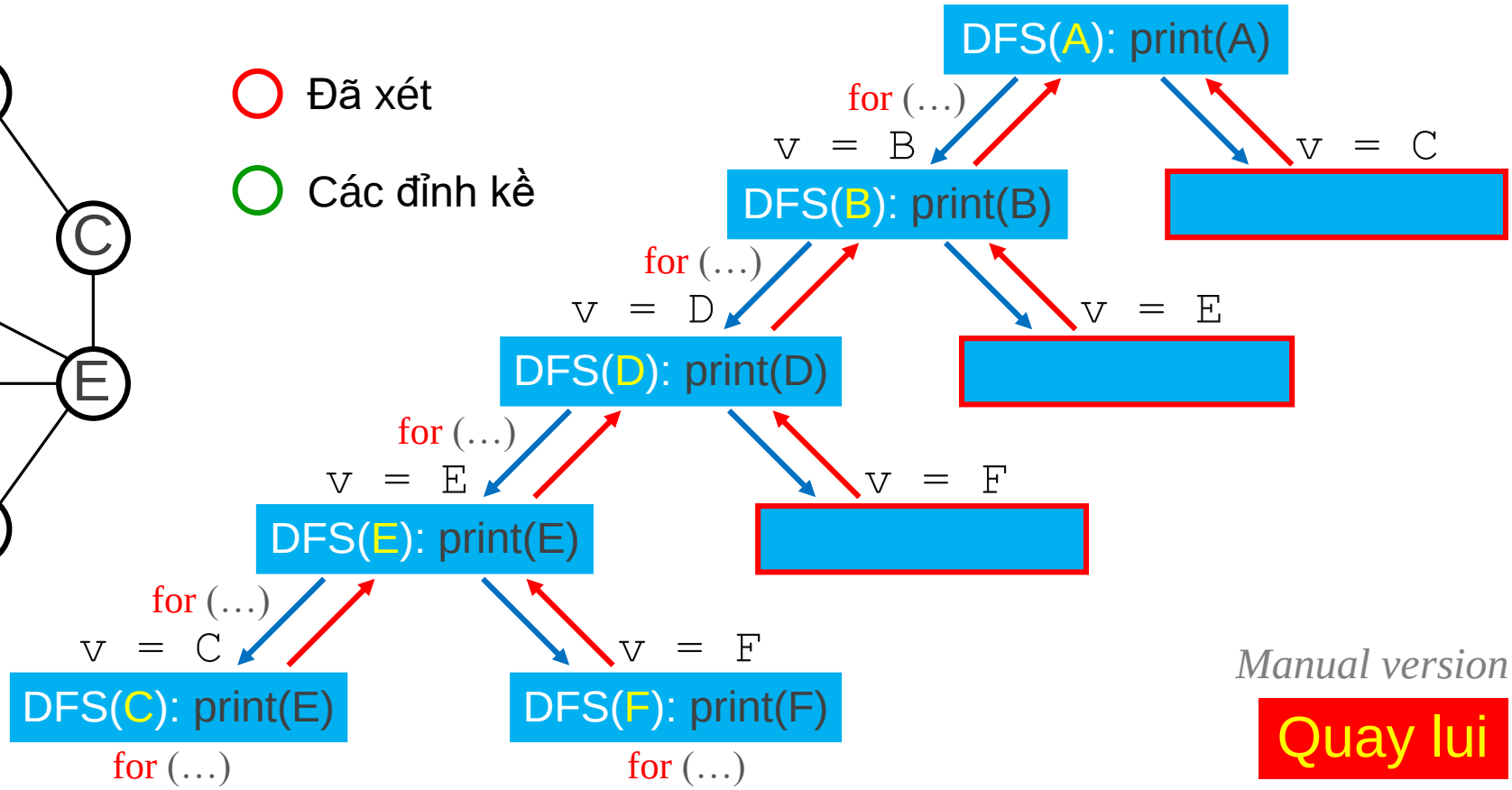
Bat dau tu dinh: A

Duyet: A B D E C F



Đã xét

Các đỉnh kề

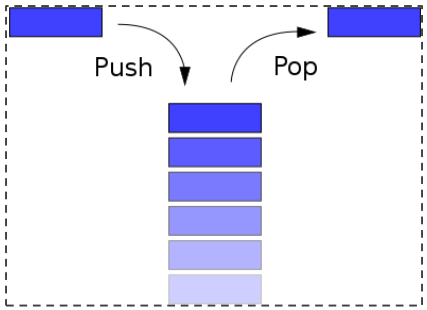


Manual version

Quay lui

DFS dùng ngăn xếp - stack khử đệ quy

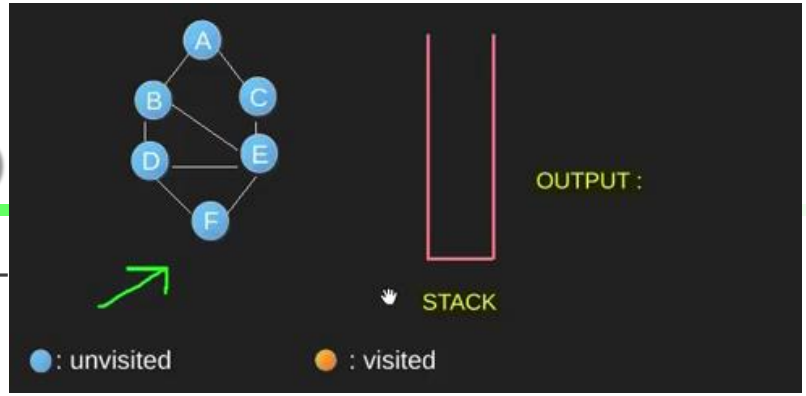
```
DFS(u){
        Bước 1: Khởi tạo
        stack = ∅;                          //khởi tạo stack là ∅
        push(stack, u);                     //đưa đỉnh u vào stack
        <Thăm đỉnh u>;                      //duyệt đỉnh u
        chuaxet[u] = false;                 //xác nhận đã duyệt u
        Bước 2: Lặp
        while(stack ≠ ∅){
                s = pop(stack);             //lấy 1 đỉnh ở đầu stack
                for(t ∈ Ke(s)){
                        if(chuaxet[t]){     //nếu chưa duyệt t
                                <Thăm đỉnh t>;      //duyệt đỉnh t
                                chuaxet[t] = false; //t đã được duyệt
                                push(stack, s);     //đưa s vào stack
                                push(stack, t);     //đưa t vào stack
                                break;              //chỉ lấy một đỉnh t
                        }
                }
        }
        Bước 3: Trả lại kết quả
        return <tập đỉnh đã duyệt>;
}
```



// Stack

// Last In First Out

DFS dùng ngăn xếp



DFS(A)

```
DFS(u){
        Bước 1: Khởi tạo
        stack = ∅;                          //khởi tạo stack là ∅
        push(stack, u);                     //đưa đỉnh u vào stack
        <Thăm đỉnh u>;                      //duyệt đỉnh u
        chuaxet[u] = false;                 //xác nhận đã duyệt u
        Bước 2: Lặp
        while(stack ≠ ∅){
                s = pop(stack);             //lấy 1 đỉnh ở đầu stack
                for(t ∈ Ke(s)){
                        if(chuaxet[t]){     //nếu chưa duyệt t
                                <Thăm đỉnh t>;      //duyệt đỉnh t
                                chuaxet[t] = false; //t đã được duyệt
                                push(stack, s);     //đưa s vào stack
                                push(stack, t);     //đưa t vào stack
                                break;              //chỉ lấy một đỉnh t
                        }
                }
        }
        Bước 3: Trả lại kết quả
        return <tập đỉnh đã duyệt>;
}
```

Độ phức tạp thuật toán DFS

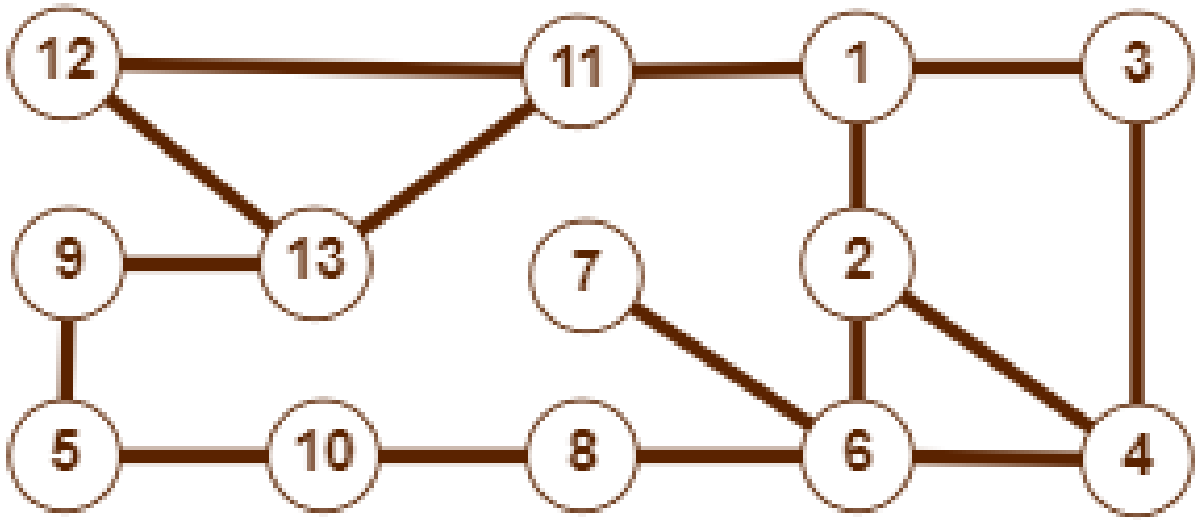
Độ phức tạp thuật toán DFS(u) phụ thuộc vào cách biểu diễn đồ thị:

- Sử dụng ma trận kề:
  - Độ phức tạp thuật toán là $O(n^2)$, n là số đỉnh
- Sử dụng danh sách cạnh:
  - Độ phức tạp thuật toán là $O(n.m)$, n là số đỉnh, m là số cạnh
- Sử dụng danh sách kề:
  - Độ phức tạp thuật toán là $O(\max(n, m))$, n là số đỉnh, m là số cạnh

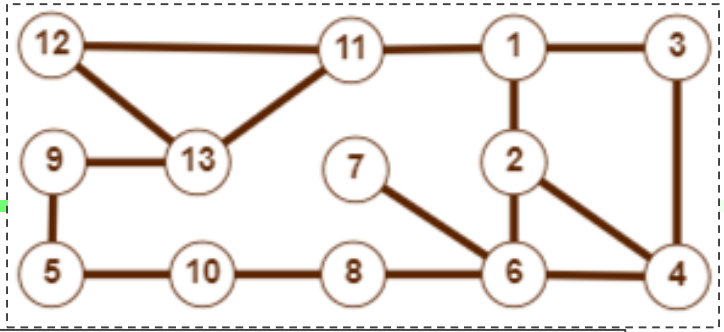
Kiểm nghiệm thuật toán DFS (1/3)

Ví dụ 1:

- Cho đồ thị gồm 13 đỉnh như hình vẽ. Hãy kiểm nghiệm thuật toán DFS(1).



Kiểm nghiệm DFS (2/3)

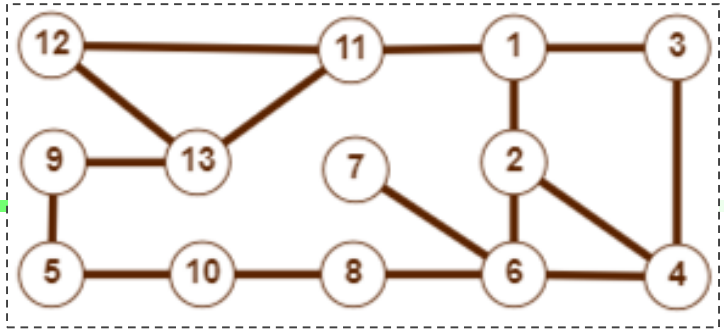


- Gọi: DFS(1)

STT	Trạng thái ngăn xếp	Danh sách đỉnh được duyệt
1	1	1
2	1, 2	1, 2
3	1, 2, 4	1, 2, 4
4	1, 2, 4, 3	1, 2, 4, 3
5	1, 2, 4	1, 2, 4, 3
6	1, 2, 4, 6	1, 2, 4, 3, 6
7	1, 2, 4, 6, 7	1, 2, 4, 3, 6, 7
8	1, 2, 4, 6	1, 2, 4, 3, 6, 7
9	1, 2, 4, 6, 8	1, 2, 4, 3, 6, 7, 8
10	1, 2, 4, 6, 8, 10	1, 2, 4, 3, 6, 7, 8, 10
11	1, 2, 4, 6, 8, 10, 5	1, 2, 4, 3, 6, 7, 8, 10, 5
12	1, 2, 4, 6, 8, 10, 5, 9	1, 2, 4, 3, 6, 7, 8, 10, 5, 9
13	1, 2, 4, 6, 8, 10, 5, 9, 13	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13
14	1, 2, 4, 6, 8, 10, 5, 9, 13, 11	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11
15	1, 2, 4, 6, 8, 10, 5, 9, 13, 11, 12	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12
16-	Lần lượt bỏ các đỉnh ra khỏi ngăn xếp	

Kết quả duyệt: 1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12

Kiểm nghiệm DFS (3/3)



- Gọi: DFS(1)

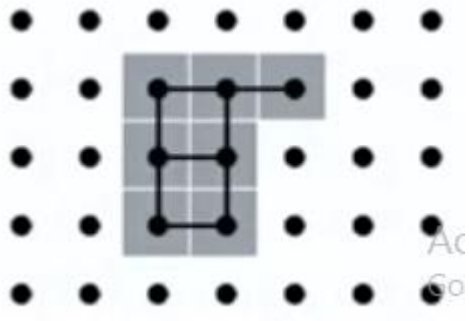
```
Dinh bat dau duyet: 1

So dinh do thi: 13
 0  1  1  0  0  0  0  0  0  0  1  0  0
 1  0  0  1  0  1  0  0  0  0  0  0  0
 1  0  0  1  0  0  0  0  0  0  0  0  0
 0  1  1  0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  1  0  0  0
 0  1  0  1  0  0  1  1  0  0  0  0  0
 0  0  0  0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0  1  0  0  0  1  0  0  0
 0  0  0  0  1  0  0  0  0  0  0  0  1
 0  0  0  0  1  0  0  1  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  1  1
 0  0  0  0  0  0  0  0  0  0  1  0  1
 0  0  0  0  0  0  0  0  0  0  1  1  0
DFS_Stack:
 1  2  4  3  6  7  8 10  5  9 13 11 12
```

```
So dinh do thi: 13
 0  1  1  0  0  0  0  0  0  0  1  0  0
 1  0  0  1  0  1  0  0  0  0  0  0  0
 1  0  0  1  0  0  0  0  0  0  0  0  0
 0  1  1  0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  1  0  0  0
 0  1  0  1  0  0  1  1  0  0  0  0  0
 0  0  0  0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0  1  0  0  0  1  0  0  0
 0  0  0  0  1  0  0  0  0  0  0  0  1
 0  0  0  0  1  0  0  1  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  1  1
 0  0  0  0  0  0  0  0  0  0  1  0  1
 0  0  0  0  0  0  0  0  0  0  1  1  0
DFS_Recursion:
 1  2  4  3  6  7  8 10  5  9 13 11 12
```

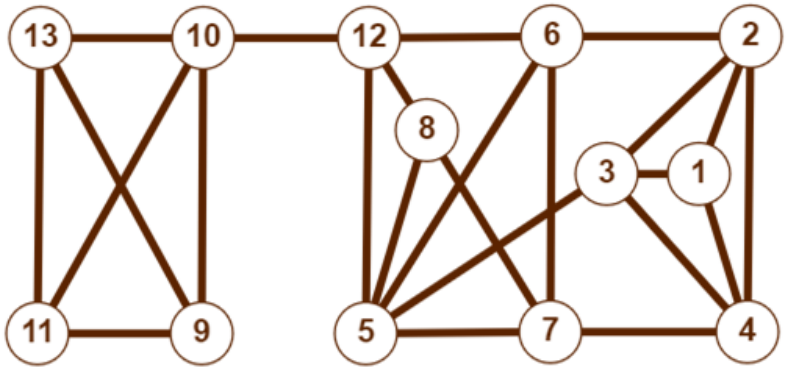
DFS: ứng dụng trong soạn thảo ảnh

- Chức năng tô màu flood fill (vd: trong Photoshop)

- Giải pháp:
  - Xây dựng đồ thị mạng lưới pixel
  - Đỉnh - pixel, cạnh - nối 2 pixel
  - DFS: nối và tô màu tất cả các pixel cùng (gần) màu



Bài tập 1



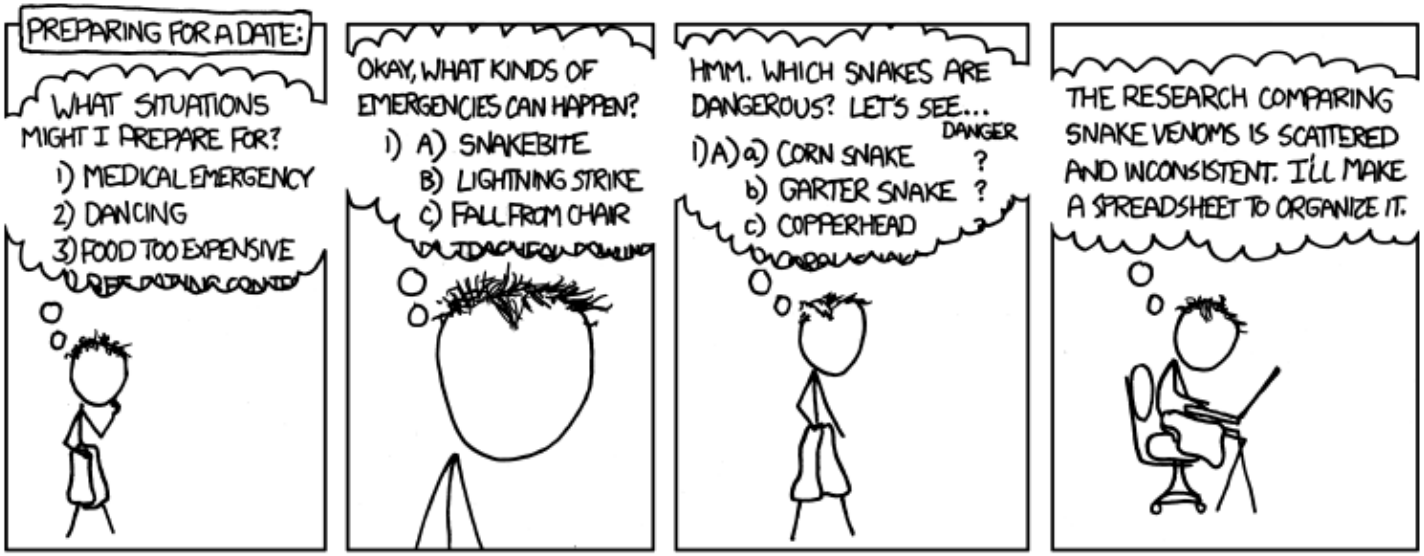
Cho đồ thị gồm 13 đỉnh được biểu diễn dưới dạng ma trận kề như hình vẽ:

- Hãy cho biết kết quả thực hiện thuật toán DFS(1).
- Chỉ rõ trạng thái của ngăn xếp và tập đỉnh được duyệt theo mỗi bước thực hiện của thuật toán.

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

(Phuong ND, 2013)

DFS, …?



Any other graph search method(s)?

⇨ BFS

Nội dung Bài 3

1. Thuật toán tìm kiếm theo chiều sâu
   Depth-First Search - DFS

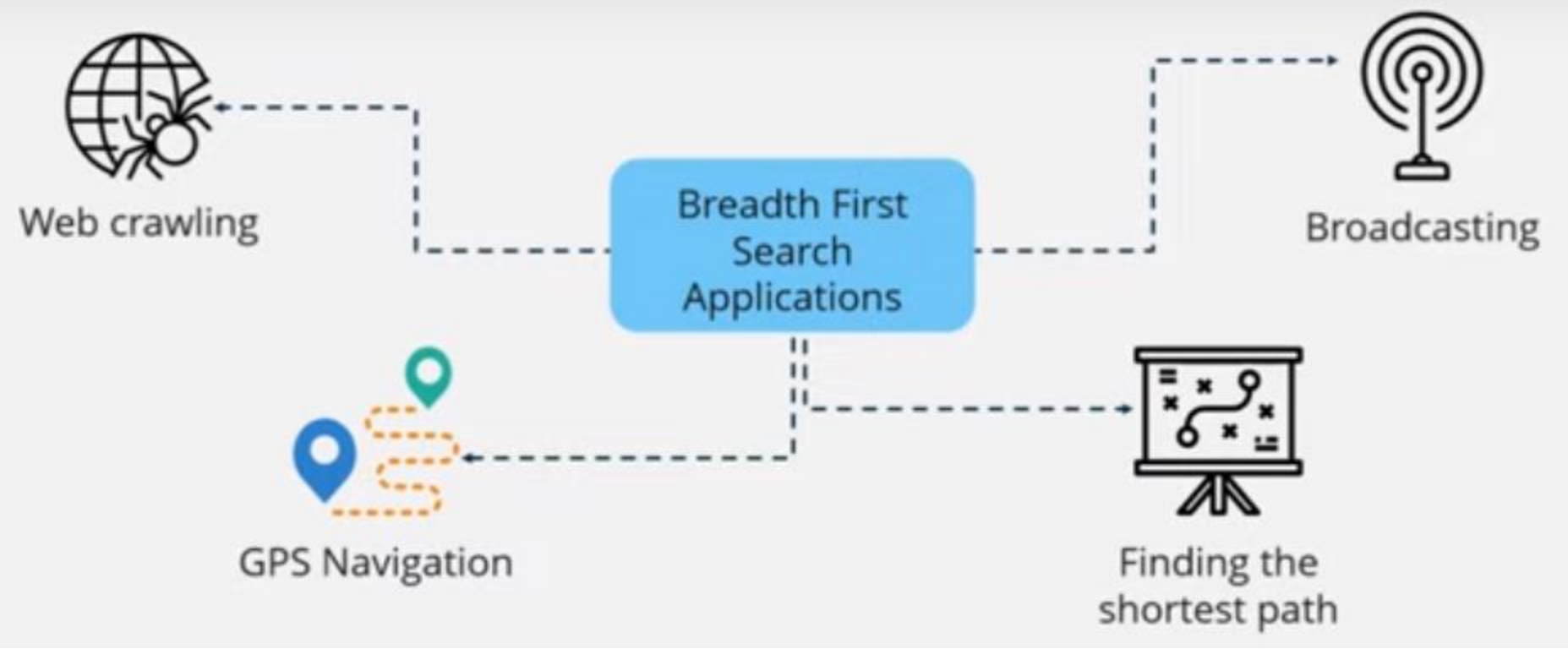
2. Thuật toán tìm kiếm theo chiều rộng
   Breadth-First Search - BFS

3. Một số ứng dụng của DFS và BFS

Tìm kiếm theo chiều rộng - BFS

Tư tưởng

- Khi tìm kiếm, ưu tiên “chiều rộng” hơn “chiều sâu”.
- Tìm kiếm xung quanh trước khi đi xuống sâu hơn.



Tìm kiếm theo chiều rộng - BFS

Thuật toán

BFS(u){

Bước 1: Khởi tạo

$queue$ = ∅; push($queue$, u); $chuaxet[u] = false$;

Bước 2: Lặp

while($queue \neq \emptyset$){

s = pop($queue$); <Thăm đỉnh s>;

for($t \in Ke(s)$){

if($chuaxet[t]$){

push($queue$, t); $chuaxet[t] = false$;

}

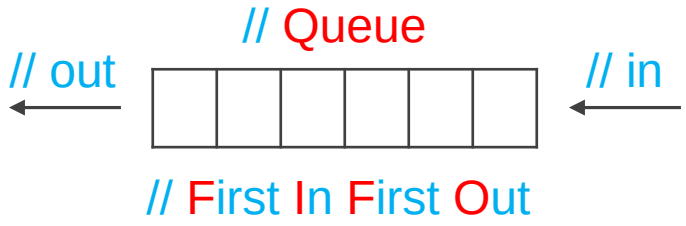
}

}

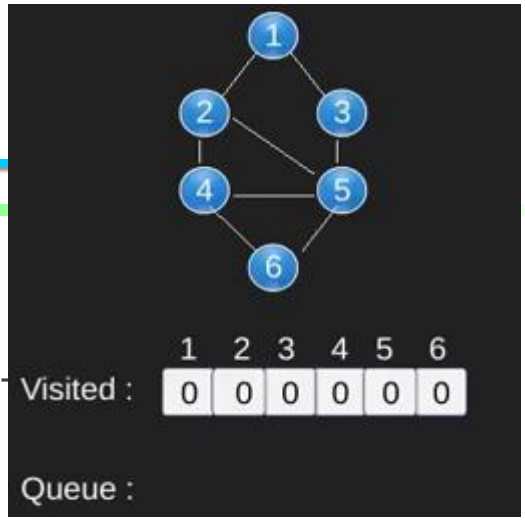
Bước 3: Trả lại kết quả

return <tập đỉnh đã duyệt>;

}



Tìm kiếm theo chiều rộng -



Thuật toán

BFS(1)

BFS(u){

Bước 1: Khởi tạo

$queue$ = ∅; push($queue$, u); $chuaxet[u] = false$;

Bước 2: Lặp

while($queue \neq \emptyset$){

s = pop($queue$); <Thăm đỉnh s>;

for($t \in Ke(s)$){

if($chuaxet[t]$){

push($queue$, t); $chuaxet[t] = false$;

}

}

}

Bước 3: Trả lại kết quả

return <tập đỉnh đã duyệt>;

}

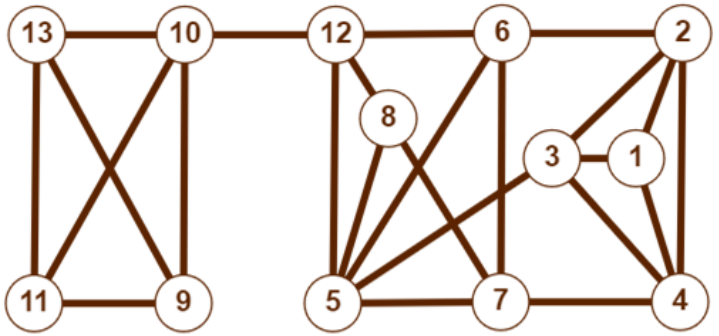
Độ phức tạp thuật toán BFS

Độ phức tạp thuật toán BFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị:

- Biểu diễn đồ thị bằng ma trận kề
  - Độ phức tạp thuật toán là $O(n^2)$, n là số đỉnh
- Biểu diễn đồ thị bằng danh sách cạnh
  - Độ phức tạp thuật toán là $O(n.m)$, n là số đỉnh, m là số cạnh
- Biểu diễn đồ thị bằng danh sách kề
  - Độ phức tạp thuật toán là $O(\max(n, m))$, n là số đỉnh, m là số cạnh

Kiểm nghiệm thuật toán BFS (1/3)

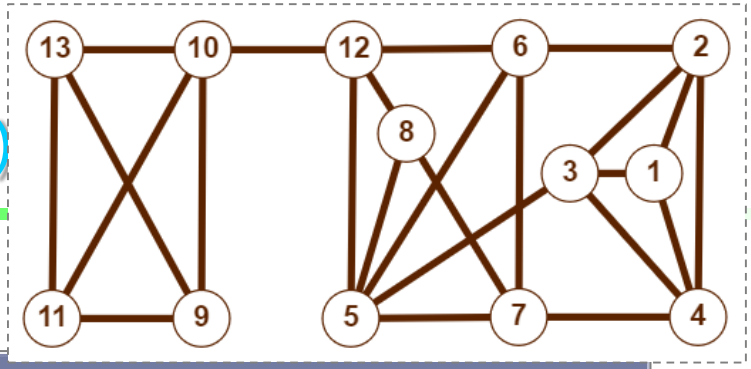
- Cho đồ thị gồm 13 đỉnh được biểu diễn dưới dạng ma trận kề như hình vẽ:
  - Hãy cho biết kết quả thực hiện thuật toán BFS(1).
  - Chỉ rõ trạng thái của hang đợi và tập đỉnh được duyệt theo mỗi bước thực hiện của thuật toán.



0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

(Phương ND, 2013)

Kiểm nghiệm BFS (2/3)

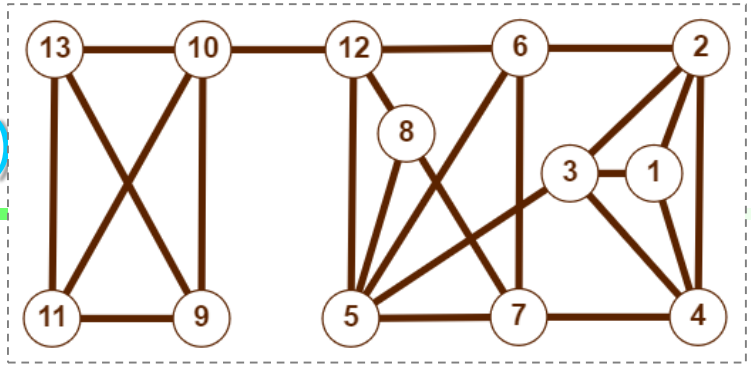


- Gọi: BFS(1)

STT	Trạng thái hàng đợi	Danh sách đỉnh được duyệt
1	1	∅
2	2, 3, 4	1
3	3, 4, 6	1, 2
4	4, 6, 5	1, 2, 3
5	6, 5, 7	1, 2, 3, 4
6	5, 7, 12	1, 2, 3, 4, 6
7	7, 12, 8	1, 2, 3, 4, 6, 5
8	12, 8	1, 2, 3, 4, 6, 5, 7
9	8, 10	1, 2, 3, 4, 6, 5, 7, 12
10	10	1, 2, 3, 4, 6, 5, 7, 12, 8
11	9, 11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10
12	11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9
13	13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11
14	∅	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11, 13

Kết quả duyệt: 1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11, 13

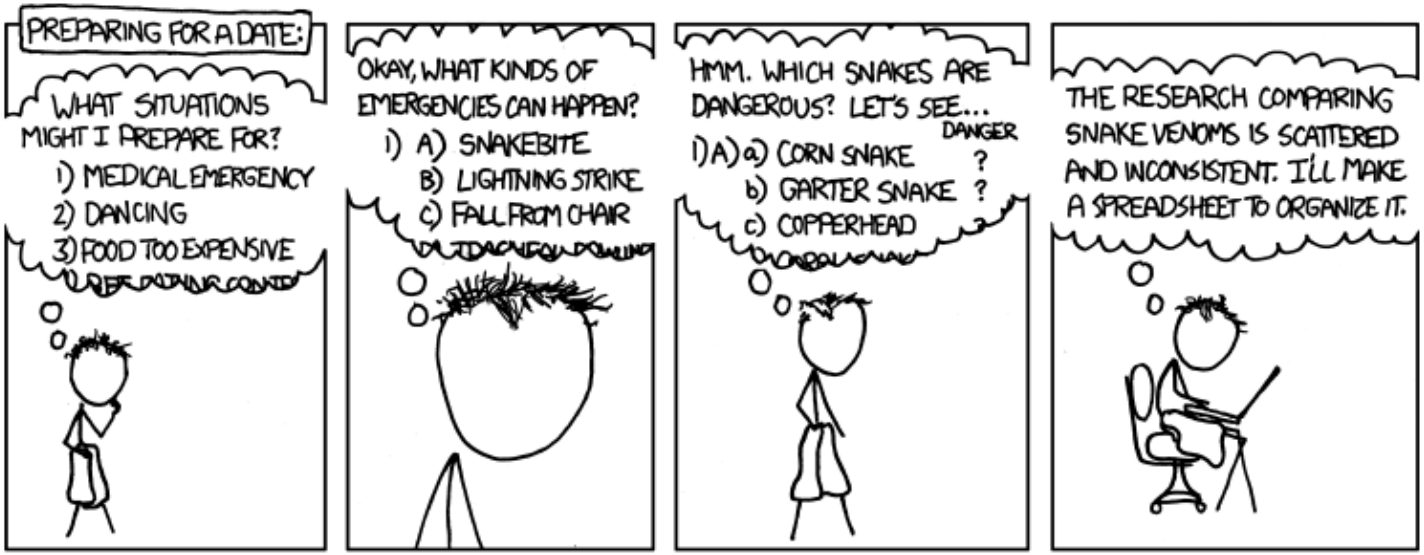
Kiểm nghiệm BFS (3/3)



- Gọi: BFS(1)

```
So dinh do thi: 13
 0  1  1  1  0  0  0  0  0  0  0  0  0
 1  0  1  1  0  1  0  0  0  0  0  0  0
 1  1  0  1  1  0  0  0  0  0  0  0  0
 1  1  1  0  0  0  1  0  0  0  0  0  0
 0  0  1  0  0  1  1  1  0  0  0  1  0
 0  1  0  0  1  0  1  0  0  0  0  1  0
 0  0  0  1  1  1  0  1  0  0  0  0  0
 0  0  0  0  1  0  1  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  1  1  0  1
 0  0  0  0  0  0  0  0  1  0  1  1  1
 0  0  0  0  0  0  0  0  1  1  0  0  1
 0  0  0  0  1  1  0  1  0  1  0  0  0
 0  0  0  0  0  0  0  0  1  1  1  0  0
Dinh bat dau duyet: 1

Ket qua:   1  2  3  4  6  5  7 12  8 10  9 11 13
--------------------------------
```

BFS



"A BFS makes a lot of sense for dating in general, actually;
it suggests dating a bunch of people casually before getting serious, rather than having a series of five-year relationships one after the other."

Chú ý

- Với đồ thị vô hướng:
  - Nếu $DFS(u) = V$ hoặc $BFS(u) = V$, ta có thể kết luận: đồ thị liên thông.
- Với đồ thị có hướng:
  - Nếu $DFS(u) = V$ hoặc $BFS(u) = V$, ta có thể kết luận: đồ thị liên thông yếu.

Nội dung Bài 3

1. Thuật toán tìm kiếm theo chiều sâu
   Depth-First Search - DFS

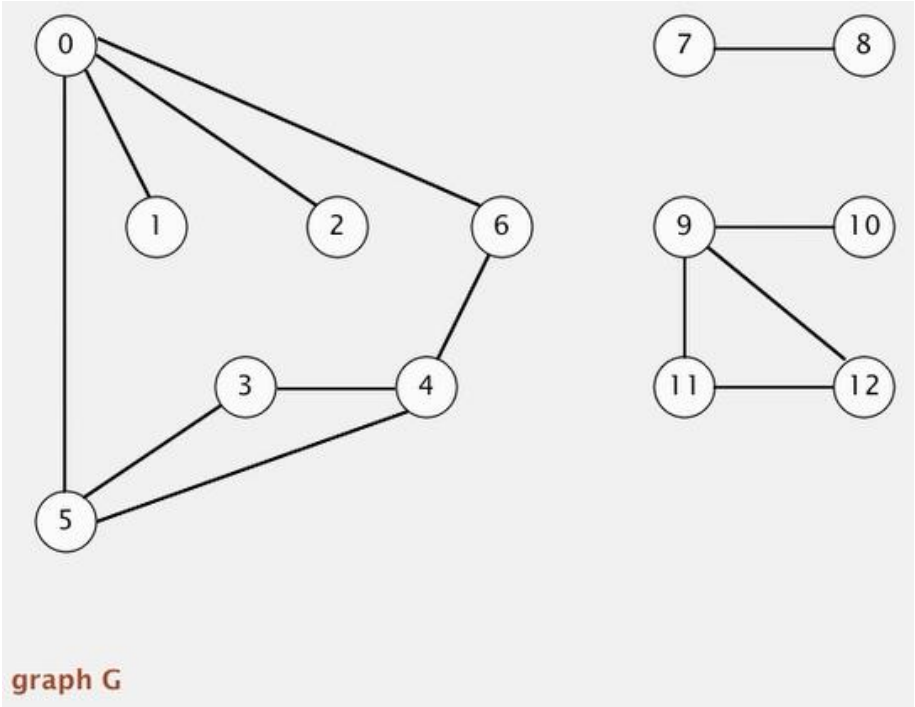
2. Thuật toán tìm kiếm theo chiều rộng
   Breadth-First Search - BFS

3. Một số ứng dụng của DFS và BFS

Xác định thành phần liên thông của đồ thị

Phát biểu bài toán:

- Cho đồ thị vô hướng $G = < V, E >$, trong đó: V là tập đỉnh, E là tập cạnh.
- Xác định các thành phần liên thông của G?
- Ví dụ:

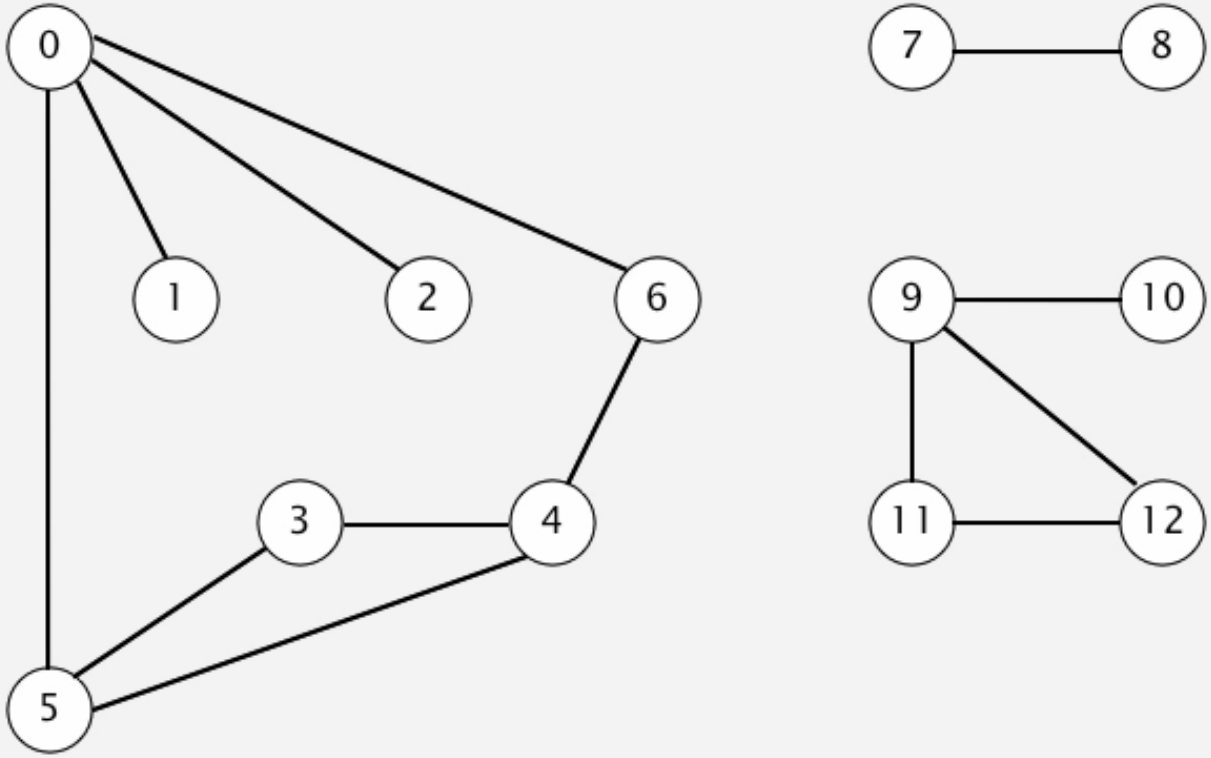


Xác định thành phần liên thông của đồ thị

Thuật toán:

```
Duyet-TPLT(){                          // duyệt thành phần liên thông
    Bước 1: Khởi tạo
    soTPLT = 0;                        // khởi tạo số thành phần liên thông = 0
    Bước 2: Lặp
    for(u ∈ V){                        // lặp trên tập đỉnh
        if(chuaxet[u]){
            soTPLT = soTPLT + 1;       // ghi nhận số TPLT
            BFS(u);                    // có thể gọi DFS(u)
            <Ghi nhận các đỉnh thuộc TPLT>;
        }
    }
    Bước 3: Trả lại kết quả
    return <các TPLT>;
}
```

Xác định thành phần liên thông của đồ thị

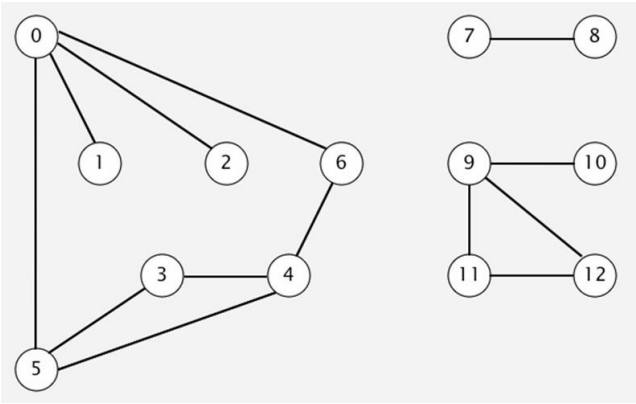


graph G

v	marked[]	id[]
0	F	–
1	F	–
2	F	–
3	F	–
4	F	–
5	F	–
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

DFS

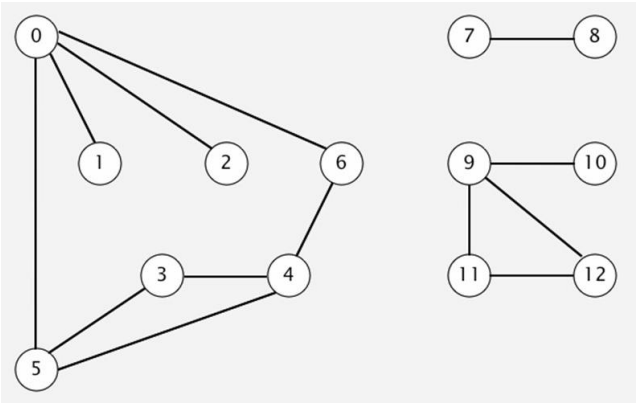
Xác định thành phần liên thông của đồ thị




```
So dinh do thi: 13
 0  1  1  0  0  1  1  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  1  1  0  0  0  0  0  0  0
 0  0  0  1  0  1  1  0  0  0  0  0  0
 1  0  0  1  1  0  0  0  0  0  0  0  0
 1  0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  1  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  1  1
 0  0  0  0  0  0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0  0  1  0  0  1
 0  0  0  0  0  0  0  0  0  1  0  1  0
TP. lien thong theo DFS 1:  0  1  2  5  3  4  6
TP. lien thong theo DFS 2:  7  8
TP. lien thong theo DFS 3:  9 10 11 12
```

DFS

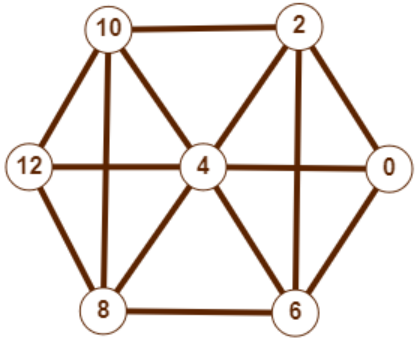
Xác định thành phần liên thông của đồ thị

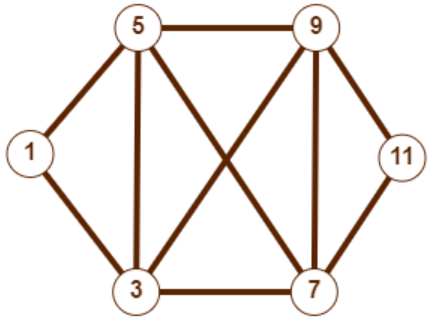



```
So dinh do thi: 13
 0  1  1  0  0  1  1  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  1  1  0  0  0  0  0  0  0
 0  0  0  1  0  1  1  0  0  0  0  0  0
 1  0  0  1  1  0  0  0  0  0  0  0  0
 1  0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  1  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  1  1
 0  0  0  0  0  0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0  0  1  0  0  1
 0  0  0  0  0  0  0  0  0  1  0  1  0
TP. lien thong theo BFS 1:  0  1  2  5  6  3  4
TP. lien thong theo BFS 2:  7  8
TP. lien thong theo BFS 3:  9 10 11 12
```

BFS

Bài tập 2





Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như hình vẽ:

- Xác định các thành phần liên thông của đồ thị?

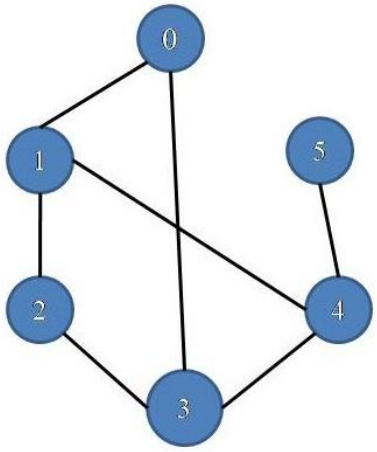
0	0	1	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	0	0
1	0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	0	1	0	1	0	0	0
1	0	1	0	1	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0	1	0	1	0
0	0	0	0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	1	0	1	0	0	0	1	0
0	0	1	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	1	0	0

(Phương ND, 2013)

Tìm đường đi giữa các đỉnh trên đồ thị (1/6)

Phát biểu bài toán:

- Cho đồ thị $G =< V, E >$ (vô hướng hoặc có hướng), trong đó V là tập đỉnh, E là tập cạnh.
- Hãy tìm đường đi từ $s \in V$ đến $t \in V$?



1. $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$
2. $0 \rightarrow 1 \rightarrow 4$
3. $0 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 4$
4. $0 \rightarrow 3 \rightarrow 4$

Tìm đường đi giữa các đỉnh trên đồ thị (2/6)

Mô tả thuật toán:

- Nếu $t \in DFS(s)$ hoặc $t \in BFS(s)$ thì ta có thể kết luận có đường đi từ s đến t trên đồ thị, ngược lại nếu $t \notin DFS(s)$ và $t \notin BFS(s)$ sẽ không có đường đi.
- Để lưu đường đi: dùng mảng $truoc[]$ với n phần tử ($n = |V|$)
  - Khởi tạo ban đầu $truoc[u] = 0$ với mọi $u \in V$.
  - Mỗi khi đưa $v \in Ke(u)$ vào ngăn xếp (nếu sử dụng DFS) hoặc hàng đợi (nếu sử dụng BFS) ta ghi nhận $truoc[v] = u$.
  - Nếu DFS & BFS không duyệt được đến t, khi đó $truoc[t] = 0$, ta kết luận không có đường đi từ s đến t.

Tìm đường đi giữa các đỉnh trên đồ thị (3/6)

Thuật toán dùng DFS:

```
DFS(s){
    Bước 1: Khởi tạo
    stack = ∅;        push(stack, s);        chuaxet[s] = false;
    Bước 2: Lặp
    while(stack ≠ ∅){
        u = pop(stack);                          // lấy 1 đỉnh ở stack
        for(v ∈ Ke(u)){
            if(chuaxet[v]){                      // nếu chưa duyệt v
                chuaxet[v] = false;              // v đã duyệt
                push(stack, u);                  // đưa u vào stack
                push(stack, v);                  // đưa v vào stack
                truoc[v] = u;                    // lưu truoc[v] là u
                break;                           // chỉ lấy 1 đỉnh
            }
        }
    }
    Bước 3: Trả lại kết quả
    return <tập đỉnh đã duyệt>;
}
```

Tìm đường đi giữa các đỉnh trên đồ thị (4/6)

Thuật toán dùng BFS - đường đi ít cạnh nhất:

```
BFS(s){
    Bước 1: Khởi tạo
    queue = ∅;        push(queue, s);        chuaxet[s] = false;
    Bước 2: Lặp
    while(queue ≠ ∅){
        u = pop(queue);                          // lấy 1 đỉnh ở queue
        for(v ∈ Ke(u)){
            if(chuaxet[v]){                      // nếu chưa duyệt v
                push(queue, v);                  // đưa v vào queue
                chuaxet[v] = false;              // v đã duyệt
                truoc[v] = u;                    // lưu truoc[v] là u
            }
        }
    }
    Bước 3: Trả lại kết quả
    return <tập đỉnh đã duyệt>;
}
```

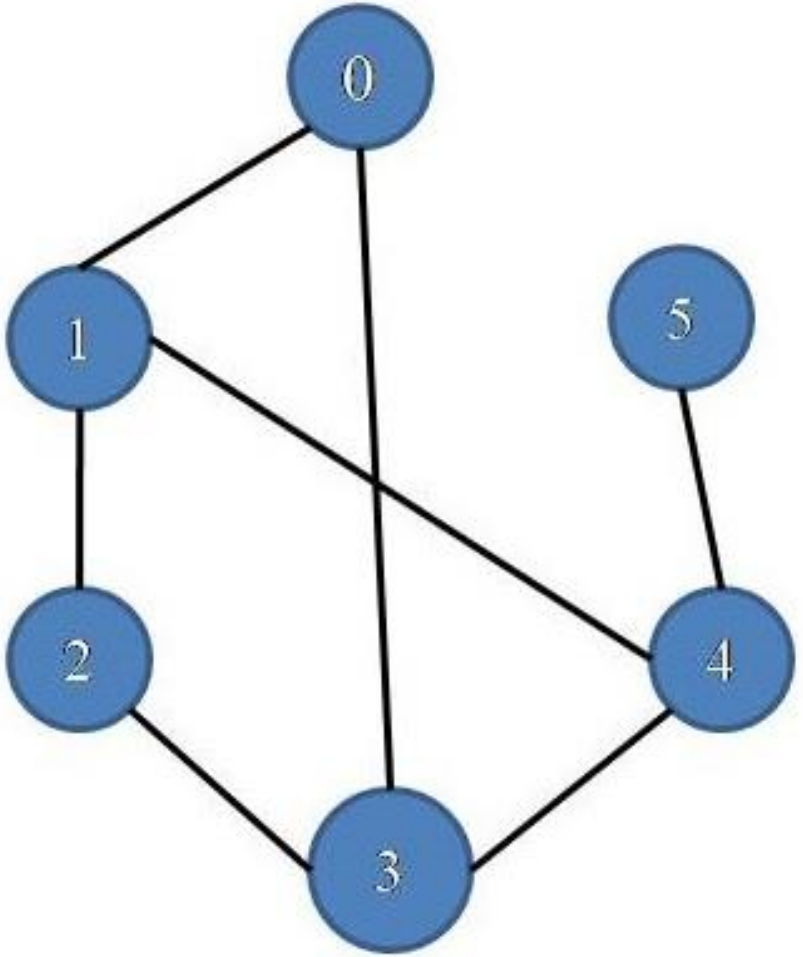
Tìm đường đi giữa các đỉnh trên đồ thị (5/6)

Ghi nhận đường đi:

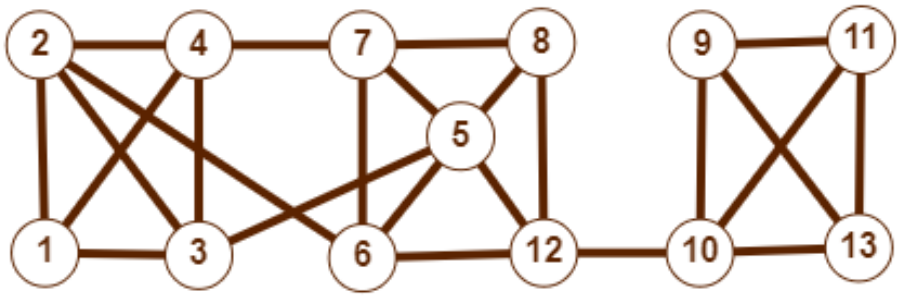
```
Ghi_Nhan_Duong_Di(s, t){
    if(truoc[t] == 0){
        <Không có đường đi từ s tới t>;
    }
    else{
        <Đưa ra đỉnh t>;            // đưa ra đỉnh t trước
        u = truoc[t];               // u là đỉnh trước khi đến được t
        while(u ≠ s){
            <Đưa ra đỉnh u>;
            u = truoc[u];           // lần ngược lại đỉnh trước u
        }
        <Đưa ra đỉnh s>;
    }
}
```

Tìm đường đi giữa các đỉnh trên đồ thị (6/6)

```
So dinh cua do thi: 6
Tim duong di tu dinh 0 den dinh 4
Danh sach canh:
  0  1
  0  3
  1  0
  1  2
  1  4
  2  1
  2  3
  3  0
  3  2
  3  4
  4  1
  4  3
  4  5
  5  4
Cac duong di khac nhau:
0  1  2  3  4
0  1  4
0  3  2  1  4
0  3  4
```



Bài tập 3



Cho đồ thị gồm 13 đỉnh được biểu diễn dưới dạng ma trận kề như ở bên cạnh.

- Tìm đường đi từ đỉnh 1 đến đỉnh 13 của đồ thị

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

(Phương ND, 2013)

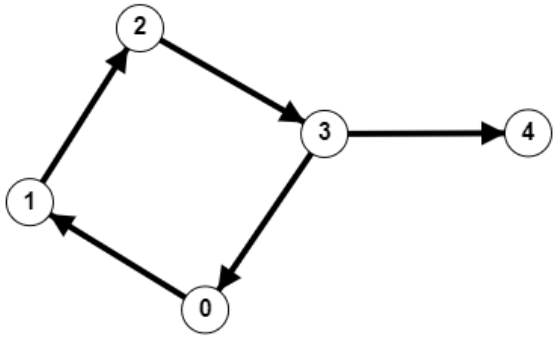
Tính liên thông mạnh trên đồ thị có hướng

- Phát biểu bài toán:
  - Đồ thị có hướng $G = < V, E >$ là liên thông mạnh nếu giữa hai đỉnh bất kỳ của nó đều tồn tại đường đi.
  - Cho trước đồ thị có hướng $G = < V, E >$. Kiểm tra xem G có liên thông mạnh hay không?
- Thuật toán:

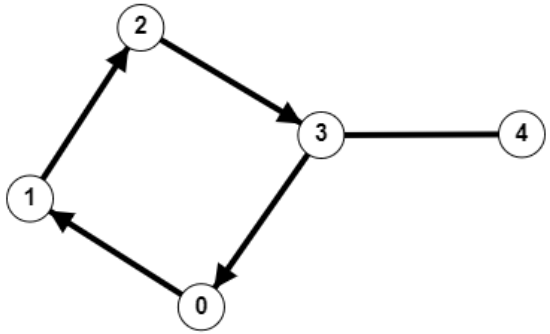
```
bool Strongly_Connected(G = <V, E>){ // kt tính liên thông mạnh của G
        ReInit( );                   // ∀u ∈ V: chuaxet[u] = true;
        for(u ∈ V){                  // lặp trên tập đỉnh
            if(BFS(u) ≠ V)           // có thể kiểm tra DFS(u) ≠ V
                return false;        // đồ thị không liên thông mạnh
            else
                ReInit();            // khởi tạo lại mảng chuaxet[]
        }
        return true;                 // đồ thị liên thông mạnh
}
```

Tính liên thông mạnh trên đồ thị có hướng

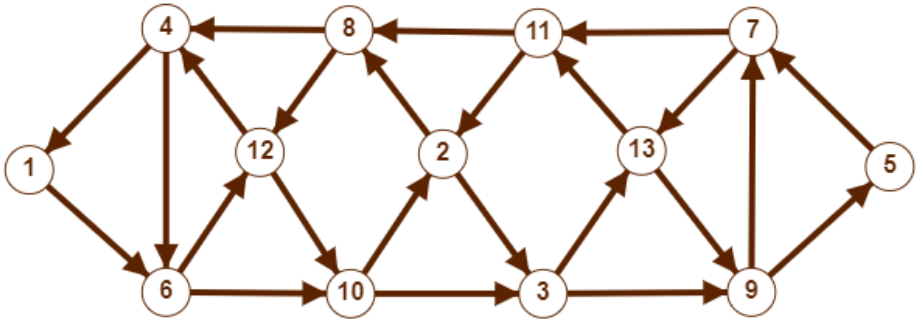
```
So dinh do thi: 5
Ma tran ke:
 0  1  0  0  0
 0  0  1  0  0
 0  0  0  1  0
 1  0  0  0  1
 0  0  0  0  0

Do thi khong lien thong manh.
```




```
So dinh do thi: 5
Ma tran ke:
 0  1  0  0  0
 0  0  1  0  0
 0  0  0  1  0
 1  0  0  0  1
 0  0  0  1  0

Do thi lien thong manh.
```



Bài tập 4



Cho đồ thị có hướng $G = < V, E >$ được biểu diễn dưới dạng ma trận kề như hình bên.

- Xác định xem G có liên thông mạnh hay không? (sử dụng thuật toán trong bài)

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

(Phương ND, 2013)

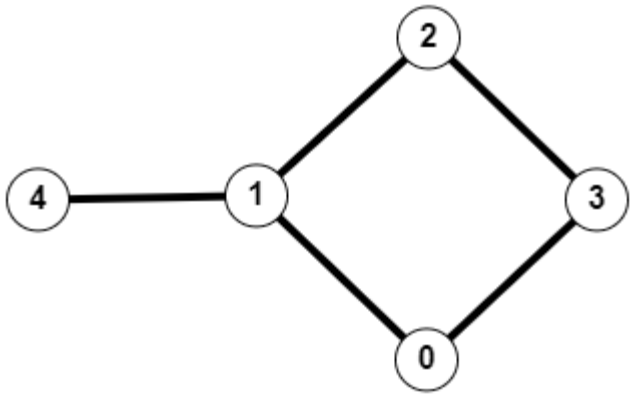
Duyệt các đỉnh trụ

- Phát biểu bài toán:
  - Đỉnh $u \in V$ của đồ thị vô hướng $G = < V, E >$ được gọi là trụ nếu loại bỏ đỉnh u cùng với các cạnh nối với u làm tăng thành phần liên thông của G. Cho trước đồ thị vô hướng (liên thông) $G = <V, E >$, tìm các đỉnh trụ của G?
- Thuật toán:

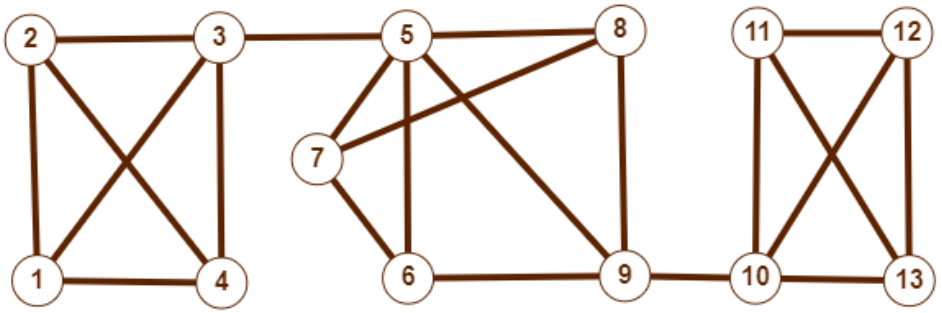
```
Duyet_Tru(G = < V, E >){
        ReInit( );                          // ∀u ∈ V: chuaxet[u] = true;
        for(u ∈ V){                         // lấy mỗi đỉnh u
            chuaxet[u] = false;             // cấm BFS hoặc DFS duyệt u
            if(BFS(v) ≠ V\{u})              // có thể kiểm tra DFS(v) ≠ V\{u}
                < u là trụ>;                // với v bất kỳ, v ≠ u
            ReInit();                       // khởi tạo lại mảng chuaxet[ ]
        }
}
```

Duyệt các đỉnh trụ

```
So dinh do thi: 5
 0  1  0  1  0
 1  0  1  0  1
 0  1  0  1  0
 1  0  1  0  0
 0  1  0  0  0

Dinh 1 la tru.
```



Bài tập 5



Cho đồ thị vô hướng $G = < V, E >$ được biểu diễn dưới dạng ma trận kề như hình bên.

- Tìm các đỉnh trụ của G?
  (sử dụng thuật toán trong bài)

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

(Phương ND, 2013)

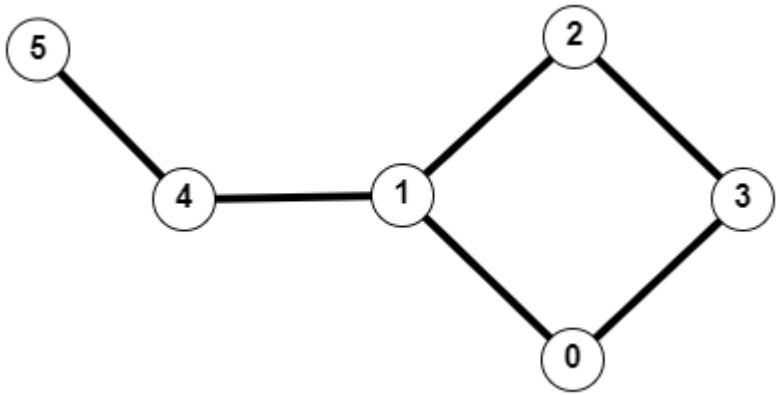
Duyệt các cạnh cầu

- Phát biểu bài toán:
  - Cạnh $e \in E$ của đồ thị vô hướng $G = < V, E >$ được gọi là cạnh cầu nếu loại e làm tăng thành phần liên thông của G. Cho trước đồ thị vô hướng (liên thông) $G = < V, E >$, tìm các cạnh cầu của G?
- Thuật toán:

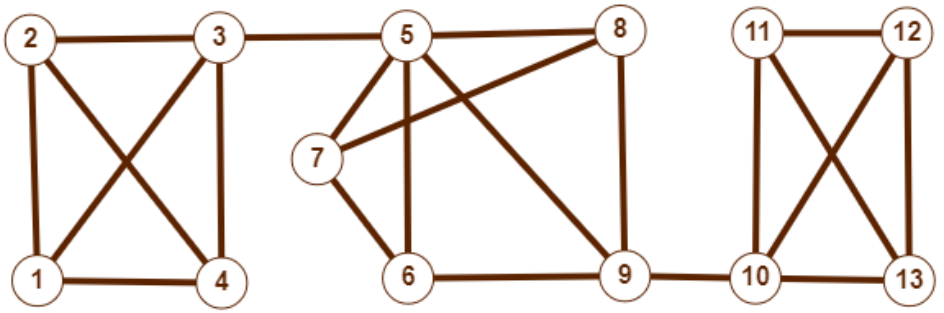
```
Duyet_Cau(G = < V, E >){
        ReInit( );                      // ∀u ∈ V: chuaxet[u] = true;
        for(e ∈ E){                     // lấy mỗi cạnh của đồ thị
            E = E \ {e};                // loại bỏ cạnh e ra khỏi đồ thị
            if(BFS(1) ≠ V)              // có thể kiểm tra DFS(1) ≠ V
                < e là cầu>;
            E = E ∪ {e};                // hoàn trả lại cạnh e
            ReInit();                   // khởi tạo lại mảng chuaxet[ ]
        }
}
```

Duyệt các cạnh cầu

```
So dinh do thi: 6
 0  1  0  1  0  0
 1  0  1  0  1  0
 0  1  0  1  0  0
 1  0  1  0  0  0
 0  1  0  0  0  1
 0  0  0  0  1  0

Canh   1   4 la canh cau.
Canh   4   5 la canh cau.
```



Bài tập 6



Cho đồ thị vô hướng $G = < V, E >$ được biểu diễn dưới dạng ma trận kề như hình bên.

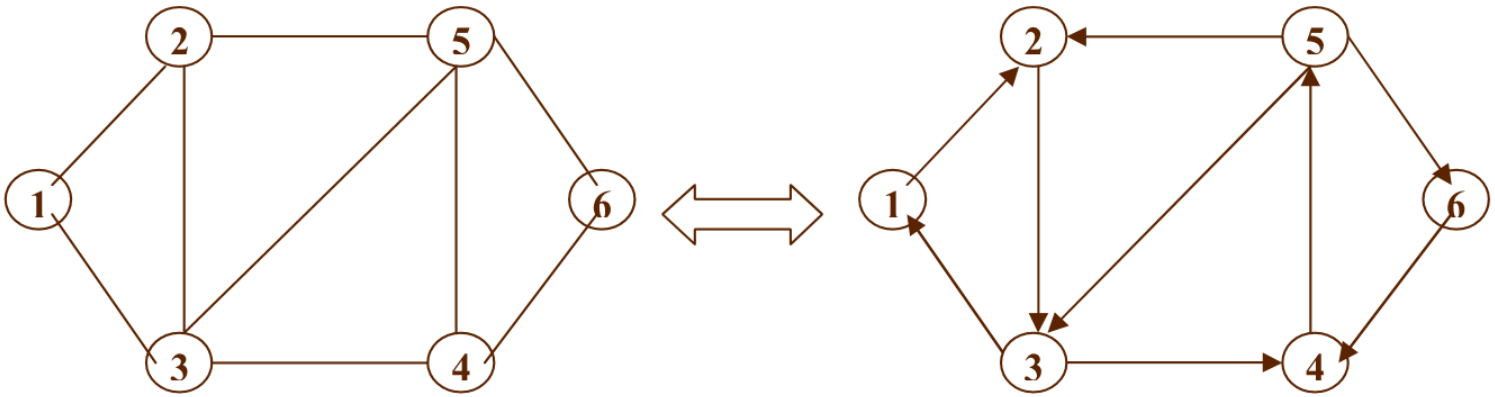
- Tìm các cạnh cầu của G ?
  (sử dụng thuật toán trong bài)

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

(Phương ND, 2013)

Bài toán định chiều đồ thị (1/2)

- Định nghĩa:
  - Phép định chiều đồ thị vô hướng liên thông là phép biến đổi đồ thị vô hướng liên thông thành đồ thị có hướng liên thông mạnh.
  - Đồ thị vô hướng $G = < V, E >$ được gọi là đồ thị định chiều được nếu có thể dịch chuyển được thành đồ thị có hướng liên thông mạnh bằng cách định chiều mỗi cạnh vô hướng thành một cung có hướng.
- Ví dụ



(Phương ND, 2013)

Bài toán định chiều đồ thị (2/2)

- Định lý:
  - Đồ thị vô hướng liên thông $G = < V, E >$ định chiều được khi và chỉ khi tất cả các cạnh $e \in E$ của G đều không phải là cầu.
- Một số vấn đề cần quan tâm:
  - Chứng minh một đồ thị vô hướng là định chiều được.
  - Viết chương trình kiểm tra một đồ thị vô hướng có định chiều được hay không?
  - Chỉ ra một phép định chiều trên một đồ thị vô hướng.

Tóm tắt

1. Thuật toán duyệt đồ thị theo chiều sâu bắt đầu tại đỉnh $u \in V$: $DFS(u)$
2. Thuật toán duyệt đồ thị theo chiều rộng bắt đầu tại đỉnh $u \in V$: $BFS(u)$
3. Ứng dụng các thuật toán $DFS(u)$ và $BFS(u)$:
   - Duyệt tất cả các đỉnh của đồ thị;
   - Duyệt tất cả các thành phần liên thông của đồ thị;
   - Tìm đường đi từ đỉnh s đến đỉnh t trên đồ thị;
   - Kiểm tra tính liên thông mạnh của đồ thị;
   - Duyệt các đỉnh trụ của đồ thị;
   - Duyệt các cạnh cầu của đồ thị;
   - Kiểm tra một đồ thị có định chiều được hay không?

Bài tập

- Cài đặt các thuật toán đã học dựa theo hướng dẫn trong giáo trình;
- Làm các bài tập trong slide bài giảng (download theo link đã được cung cấp);
- Làm các bài tập 1 – 12, Bài tập Chương 3 trong giáo trình.

Kết thúc Bài 3

- Câu hỏi và thảo luận?