Giới thiệu về đồ thị

Giảng viên: Tạ Việt Cường
Phòng HMI – Khoa CNTT

Đồ thị

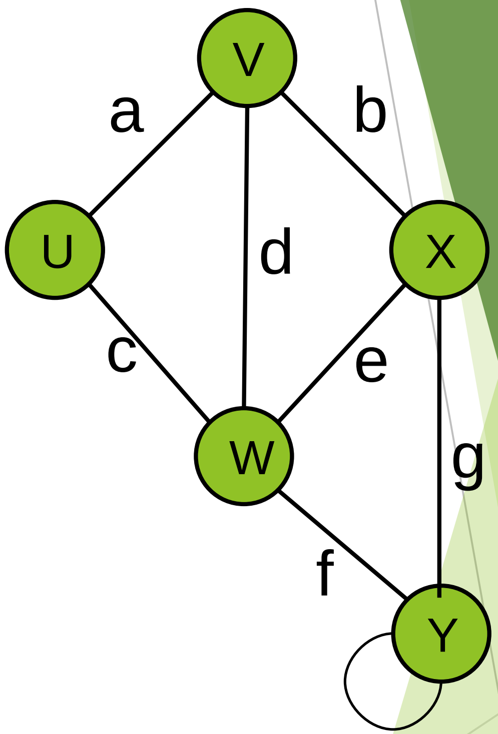
- Đồ thị gồm có <V, E>: Đỉnh và cạnh
  - V - Đỉnh
  - E - cạnh
- Ví dụ:
  - Mạng xã hội facebook:
    - Đỉnh là users
    - Cạnh là u có là bạn của v hay ko
  - Đường giao thông
    - Đỉnh là các giao điểm
    - Cạnh là các đường

Đồ thị vô hướng và có hướng

- Cạnh vô hướng - undirected edges:
  - a là friend của b <-> b là friend của a
- Cạnh có hướng - directed edges:
  - học tin 4 -> học LTNC -> học DSA
- Đồ thị vô hướng - undirected graph:
  - Gồm các cạnh vô hướng
- Đồ thị có hướng - directed graph:
  - Gồm các cạnh vô hướng và có hướng

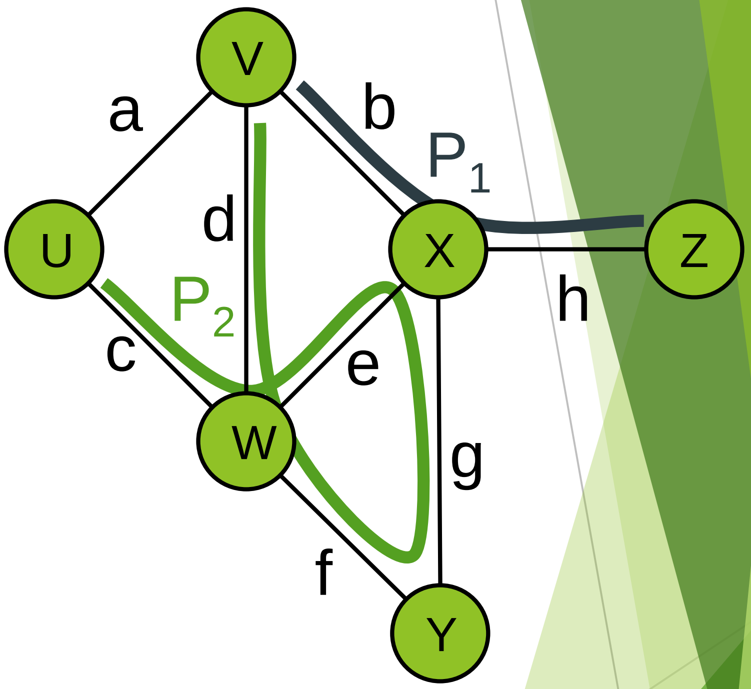
Các khái niệm cơ bản

- Đỉnh và endpoints:
  - Cạnh a có 2 endpoints là U và V
- 2 đỉnh U, V kề nhau nếu có cạnh nối giữa chúng
- Bậc của đỉnh:
  - Số cạnh nối với nó
  - Bậc vào/ra trong trường hợp đồ thị có hướng
- Y có 1 self-loop



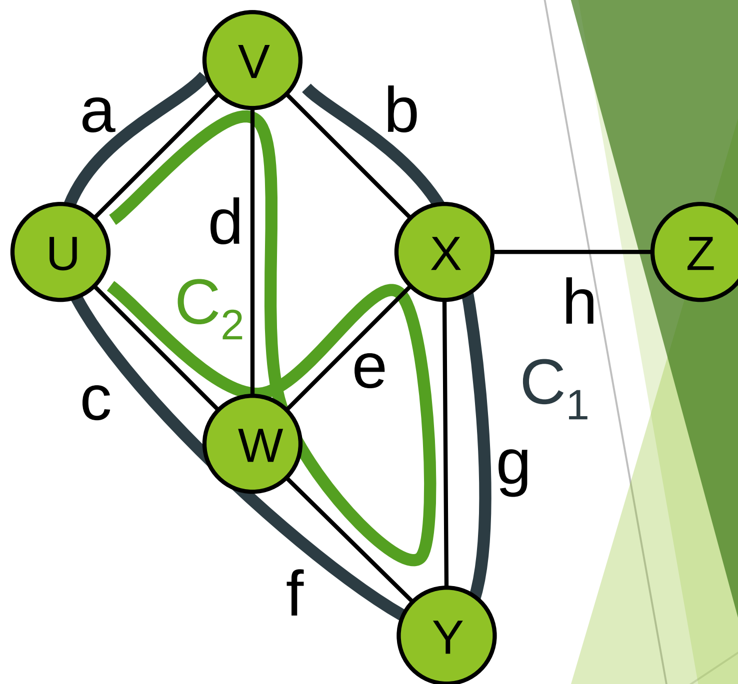
Các khái niệm cơ bản

- Đường đi:
  - Dãy các đỉnh và cạnh kề nhau
- Đường đi đơn:
  - Không lặp lại đỉnh và cạnh
- Ví dụ:
  - P_1=(V,b,X,h,Z) là đường đi đơn
  - P_2=(U,c,W,e,X,g,Y,f,W,d,V): lặp lại đỉnh W

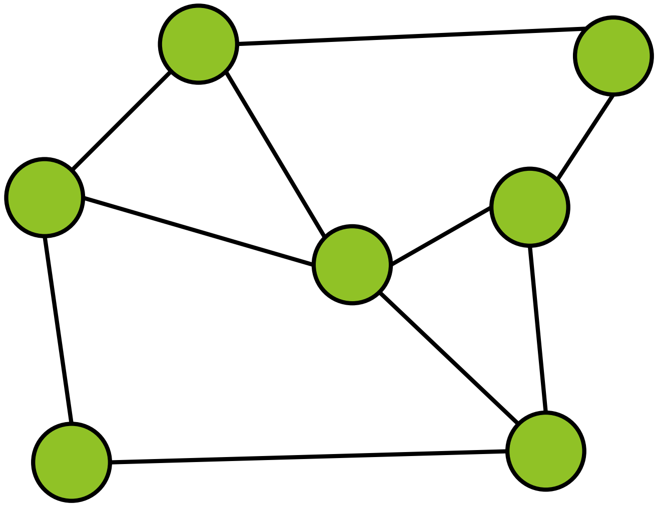


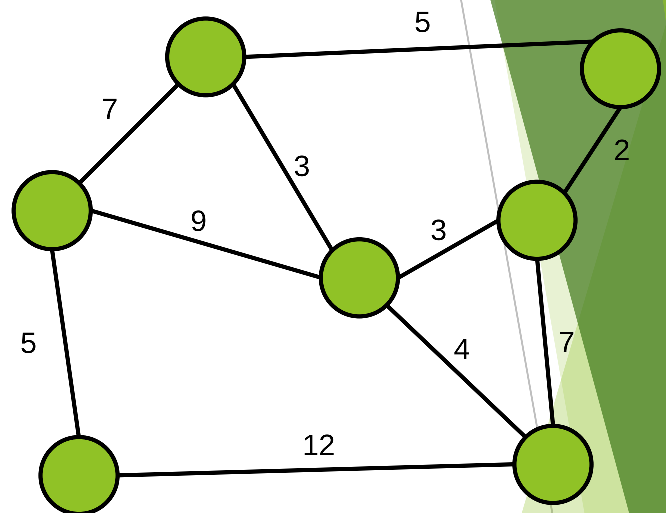
Các khái niệm cơ bản

- Chu trình:
  - Đường đi khép kín
- Chu trình đơn:
  - Không có đỉnh lặp lại trừ đỉnh đầu và đỉnh cuối
- Ví dụ:
  - C_1=(V,b,X,g,Y,f,W,c,U,a,↵)
  - C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,↵)



Đồ thị không trọng số và có trọng số





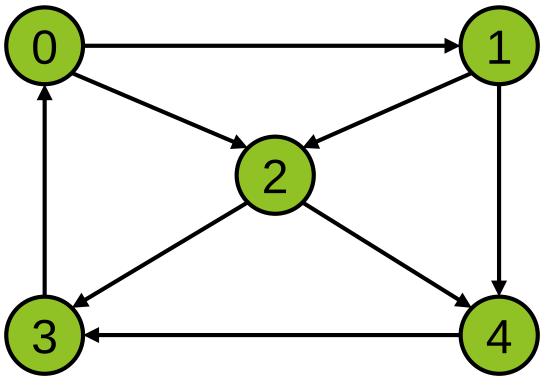
- G = <V, E>
  - Cạnh (u, v) thuộc E đc gắn kèm trọng số
- Ví dụ:
  - Friends:
  - Roadmap
- Đồ thị không trọng số <-> Độ thị trọng số cạnh bằng 1

Biểu diễn đồ thị

- G = (V, E); V = {0, 1,…, n-1}
  - Đánh số các đỉnh từ 0 đến n-1
- Sử dụng ma trận kề A như sau

  A[u][v] = 1   if (u,v) ∈ E

  A[u][v] = 0   Otherwise



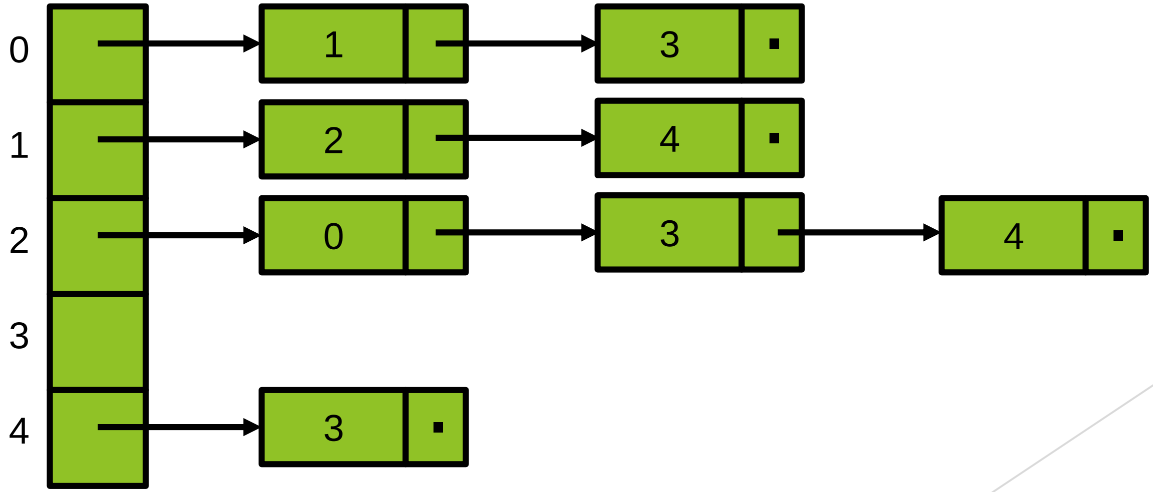
	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	1
3	1	0	0	0	0
4	0	0	0	1	0

Biểu diễn đồ thị

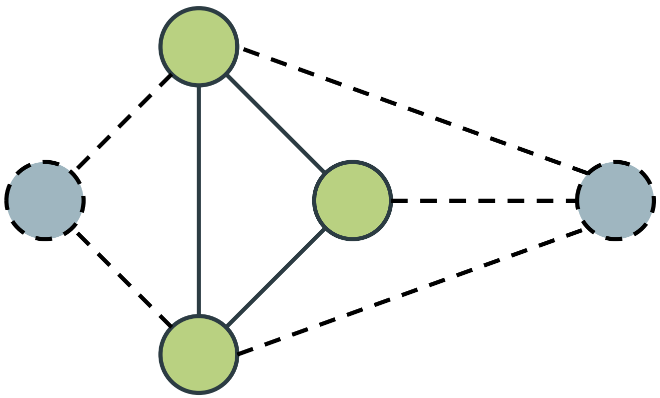
- Sử dụng danh sách vector:

G = (V, E); V = {0, 1,…, n-1}

A[i] - vector<int>: lưu các đỉnh có cạnh với i



Subgraph - Đồ thị bộ phận

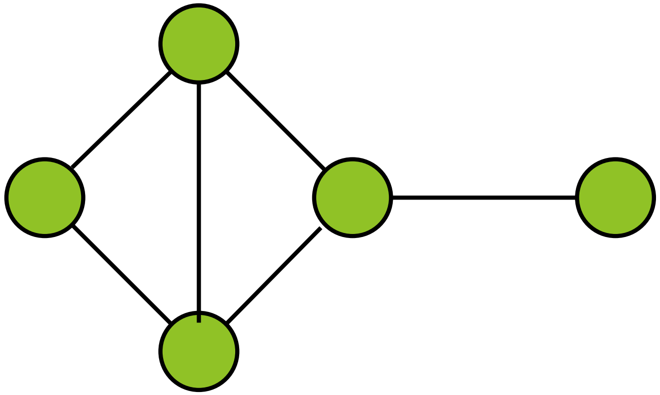


Subgraph

- spanning subgraph: chứa tất cả các đỉnh của graph

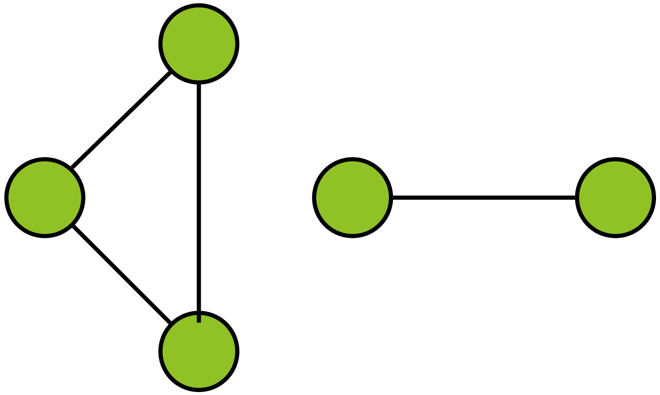
Liên thông - Connectivity

- Đồ thị gọi là liên thông nếu với mọi cặp đỉnh tồn tại đường đi giữa chúng



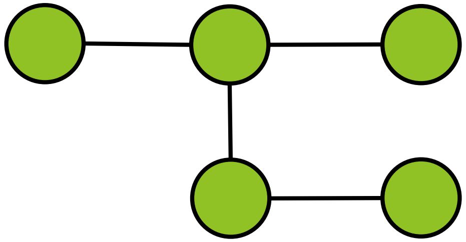
Liên thông - Connectivity

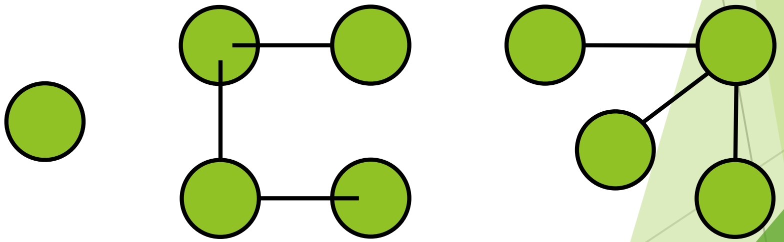
- Thành phần liên thông



Cây trong đồ thị

- Khái niệm chung về cây:
  - Đồ thị liên thông và không có chu trình
- Chú ý:Phân biệt với cây ở các bài trước (đã được chọn root)

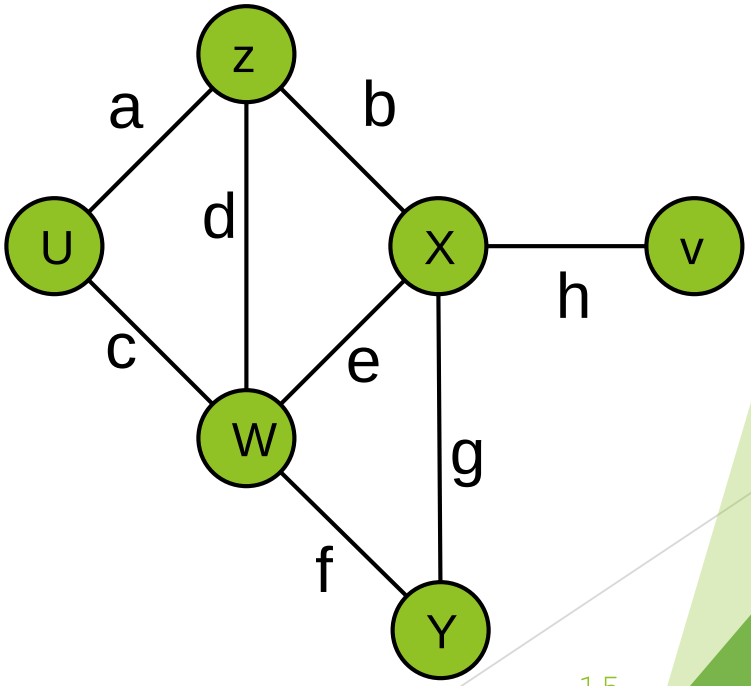

Tree


Forest

Tìm kiếm theo chiều rộng

Bài toán tìm kiếm trên đồ thị

- Cho đồ thị G = <V, E>, và 2 đỉnh u, v
  - Tìm đường đi từ u đến v
  - u - z - x - v
  - u - z - w - x - v
  - u - z - w - y - x - v



Tìm kiếm theo chiều rộng - Breadth-First Search

❖ Tìm kiếm theo chiều rộng:

Bắt đầu từ 1 đỉnh

và đi đến hết các đỉnh còn lại

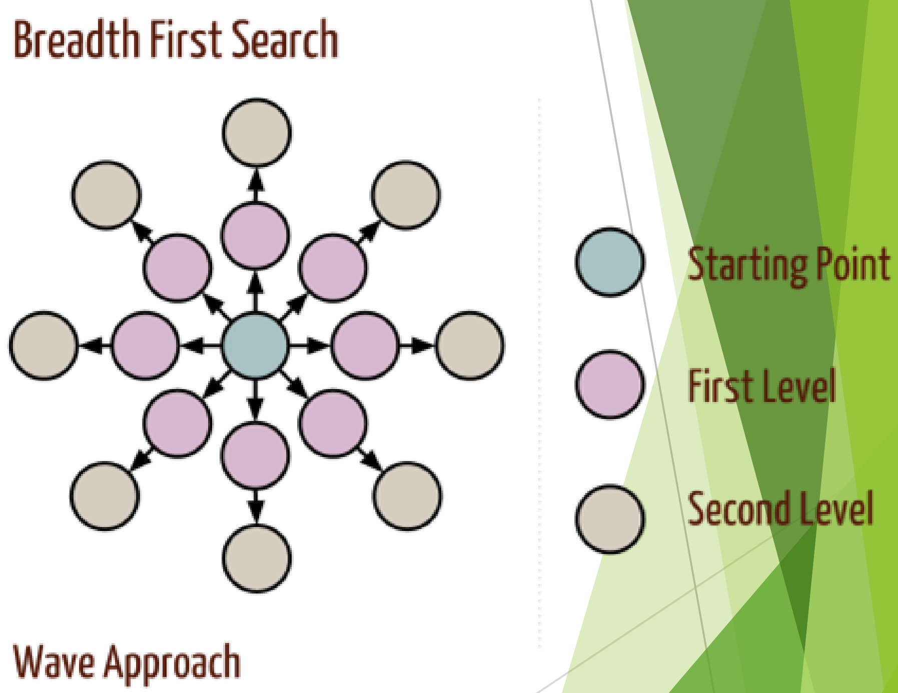
Rồi lại tiếp tục

❖ Còn gọi là loang:

Giống vết mực loang trên giấy

❖ Đường đi tìm được bởi tìm kiếm theo chiều rộng là **đường đi ngắn nhất**

Sử dụng queue Q



Pseidocode

BreadthFirstSearch (G, ***s***) {

(1) Khởi tạo ***Q rỗng***;

(3) enqueue s vào ***Q***;

(2) Gán s đã được đi qua (visited);

(4) **while** (***Q*** not empty) {

(5) ***w*** = dequeue ***Q***;

(6) **for** (với mỗi ***u*** kề với ***w***)

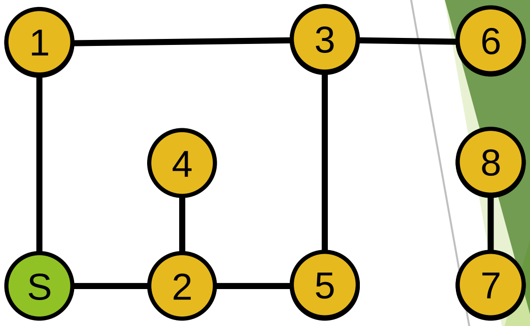
(7) **if** (***chưa thăm***) {

(8) Thêm **u** vào **Q** ;

(9) Đánh dấu u đã đi qua;

}

}



Start with all White vertices except s

Q =

Các step của BFS

❖ Sử dụng biến prev để tìm đường đi ngắn nhất

prev[S] = -1

prev[1] = S

prev[2] = S

prev[3] = -1

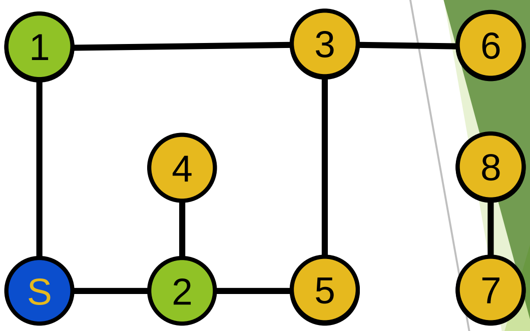
prev[4] = -1

prev[5] = -1

prev[6] = -1

prev[7] = -1

prev[8] = -1



Các step của BFS

- Sử dụng biến prev để tìm đường đi ngắn nhất

prev[S] = -1

prev[1] = S

prev[2] = S

prev[3] = 1

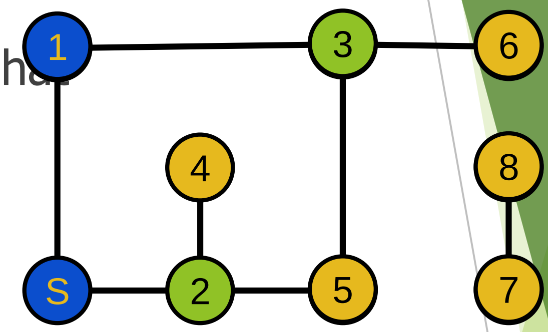
prev[4] = -1

prev[5] = -1

prev[6] = -1

prev[7] = -1

prev[8] = -1



After second time through loop

Các step của BFS

- Sử dụng biến prev để tìm đường đi ngắn nhất

prev[S] = -1

prev[1] = S

prev[2] = S

prev[3] = 1

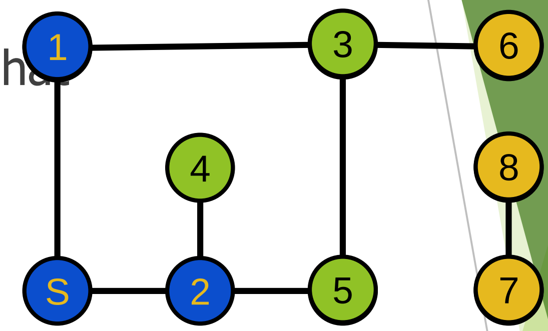
prev[4] = 2

prev[5] = 5

prev[6] = -1

prev[7] = -1

prev[8] = -1



Q =

Các step của BFS

- Sử dụng biến prev để tìm đường đi ngắn nhất

prev[S] = -1

prev[1] = S

prev[2] = S

prev[3] = 1

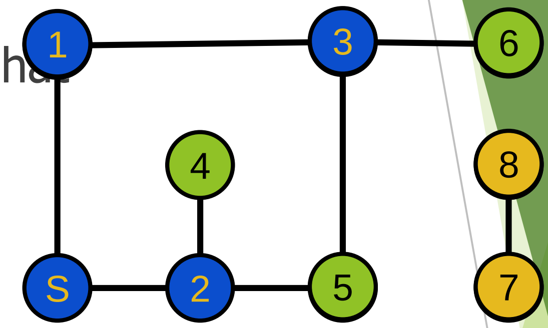
prev[4] = 2

prev[5] = 2

prev[6] = 1

prev[7] = -1

prev[8] = -1



After fourht time through loop

BFS với đồ thị có nhiều thành phần liên thông

// Travel on ***G***=(***V***, ***E***) by BFS

BreadthFirstSearch_traversal (**G**) {

for (each ***v*** ∈***V***)

mark v as unvisited;

for (each ***v*** ∈***V***)

if (***v*** not visited)

BreadthFirstSearch(***v***);

}

Complexity: BFS trên đồ thị với ***n*** đỉnh và ***m*** cạnh tốn ***O***(***n*** + ***m***)

Bài tập

1. Dùng BFS để đếm số thành phần liên thông

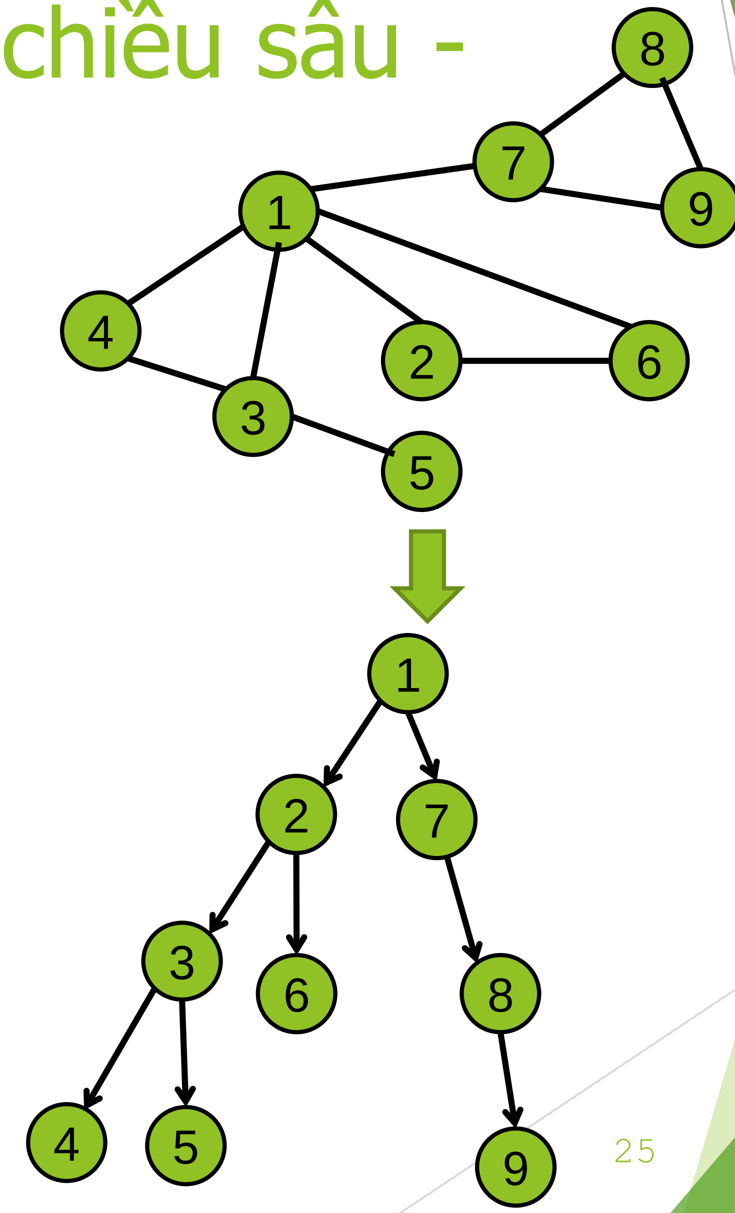
2. Cho đồ thị với các đỉnh a đến j. Và danh sách các đỉnh kề ở dưới. Tìm đường đi ngắn nhất từ a đến b

a	b	c	d	e	f	g	h	i	j
d	d	h	a	a	a	b	c	b	b
e	g		b	d	d	i		g	g
f	i		e			j			
	j		f						

Depth-First Search

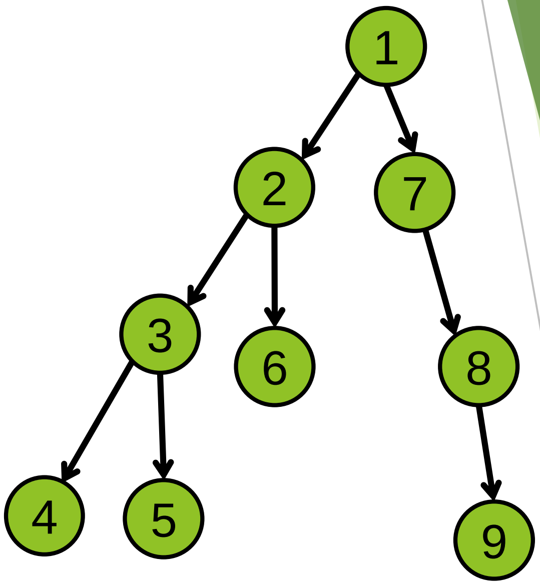
Tìm kiếm theo chiều sâu - DFS

- Đi theo 1 đỉnh cho đến khi nào ko đi được nữa
- Dùng để kiểm tra 2 đỉnh có đường đi hay không
- **KHÔNG** dùng để tìm đường đi ngắn nhất

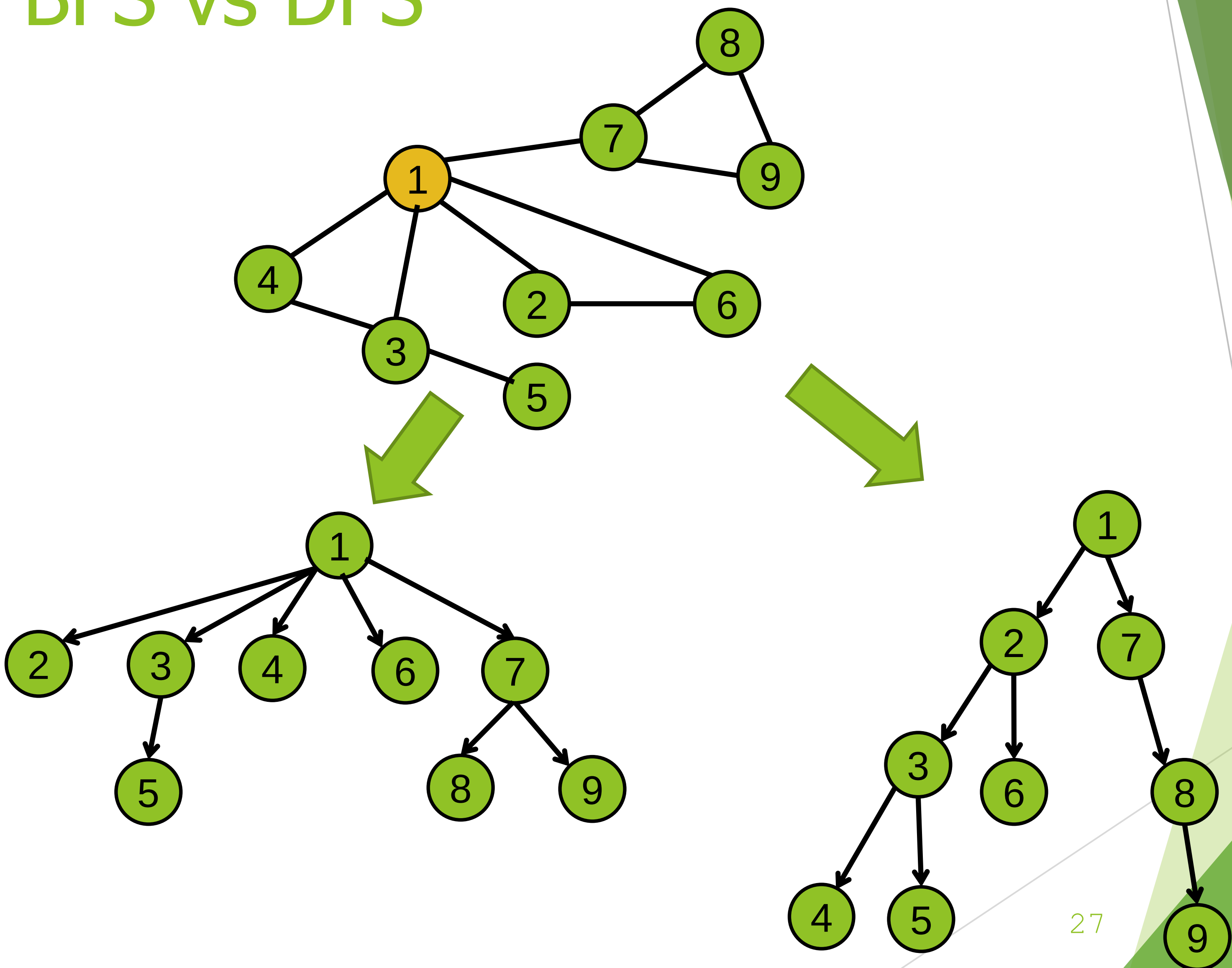


Tìm kiếm theo chiều sâu từ 1 đỉnh

```
//Depth first search from vertex v
DepthFirstSearch (v) {
      for (each u adjacent v)
        if (u not visited) {
              visit and mark u as visited;
        DepthFirstSearch (u);
        }
}
```



BFS vs DFS



Tìm kiếm chiều sâu với đồ thị nhiều thành phần liên thông

```
//Travel on G=(V, E) by DFS
DepthFirstSearch_traversal (G) {
    for (each v ∈V)
        mark v as unvisted;
    for (each v ∈V)
        if (v not visited)
            DepthFirstSearch(v);
}
```

DFS trên đồ thị với ***n*** đỉnh và ***m*** cạnh tốn ***O***(***n*** + ***m***)

Bài tập

- Sử dụng DFS để đếm số thành phần liên thông của đồ thị - Vẽ các cây DFS

a	b	c	d	e	f	g	h	i	j	k	l	m
b	a	f	b	b	c	b	b	c	a	c		g
j	d	i	h	g		e	d	k	b	i		
	e	k				m						
	g											
	h											
	j											