ĐỒ THỊ - GRAPH (tt)

DATA STRUCTURES & ALGORITHMS

ThS Nguyễn Thị Ngọc Diễm
diemntn@uit.edu.vn

NỘI DUNG

- Các khái niệm trên đồ thị
  - Định nghĩa
  - Các loại đồ thị
  - Đường đi, chu trình, liên thông
- Biểu diễn đồ thị trên máy tính
- **Các thuật toán duyệt đồ thị: BFS - DFS**
- Một số ứng dụng của tìm kiếm trên đồ thị
  - Bài toán đường đi
  - Bài toán liên thông
  - Bài toán tô màu
  - Bài toán bao đóng

Duyệt đồ thị: Graph Traversal

- Application example
  - Cho một đồ thị và đỉnh s trên đồ thị
  - Tìm tất cả các đỉnh mà s có thể đi qua
  - Tìm tất cả các đường đi (paths) từ s tới các đỉnh khác

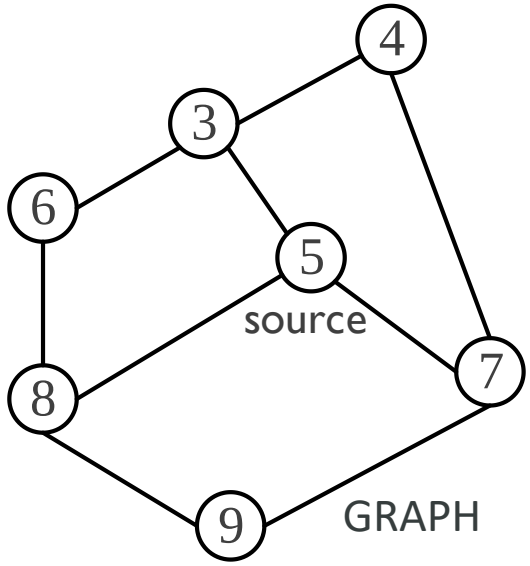
- Two common graph traversal algorithms
  - Duyệt theo chiều rộng sử dụng thuật toán Breadth-First Search (BFS)
    - Find the shortest paths in an unweighted graph
  - Duyệt theo chiều sâu sử dụng thuật toán Depth-First Search (DFS)

NỘI DUNG

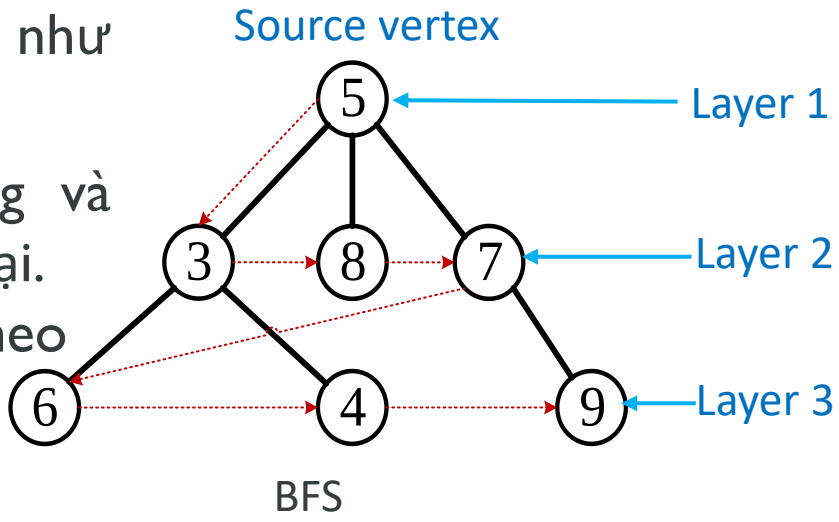
- Depth-First Search (DFS)
- **Breadth-First Search (BFS)**
  - Định nghĩa
  - Thuật toán
  - Ví dụ
  - Độ phức tạp
  - Ứng dụng
  - Nhận xét

Duyệt đồ thị theo chiều rộng

- BFS is a traversing algorithm where you should start traversing from a selected vertex (source or starting node) and traverse the graph layer-wise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

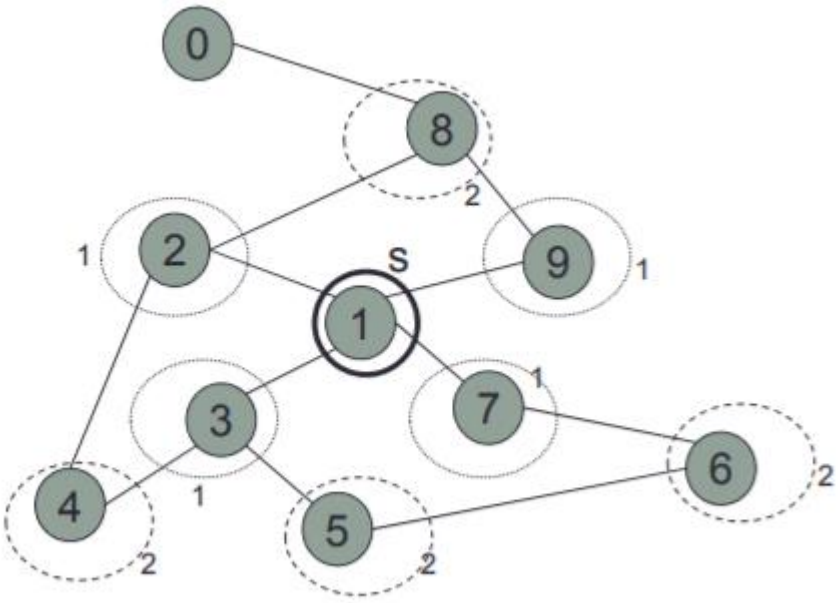


- Như tên gọi duyệt theo chiều rộng, việc duyệt đồ thị theo chiều rộng được thực hiện như sau:
  - Đầu tiên di chuyển theo chiều ngang và duyệt qua tất cả các đỉnh tại layer hiện tại.
  - Tiếp đến mới di chuyển đến layer tiếp theo



BFS and Shortest Path Problem

- Given any source vertex s, BFS visits the other vertices at **increasing distances** away from s. In doing so, BFS discovers paths from s to other vertices.
- What do we mean by "distance"? The number of edges on a path from s.



Example:

Consider s = vertex 1

Nodes at distance 1?
2, 3, 7, 9

Nodes at distance 2?
8, 6, 5, 4

Nodes at distance 3?
0

Cài đặt BSF

Để cài đặt thuật toán BSF, ta có thể sử dụng 2 cách;

1. Sử dụng hàng đợi
2. Sử dụng thuật toán loang

BSF algorithm using queue

- Bước 1: Khởi tạo:
  - Các đỉnh đều ở trạng thái chưa đánh dấu, ngoại trừ đỉnh xuất phát s là đã đánh dấu.
  - Một hàng đợi (Queue), ban đầu chỉ có một phần tử là s. Hàng đợi dùng để chứa các đỉnh sẽ được duyệt theo thứ tự ưu tiên chiều rộng
- Bước 2: Lặp các bước sau đến khi hàng đợi rỗng:
  - Lấy u khỏi hàng đợi, thông báo thăm u (Bắt đầu việc duyệt đỉnh u)
  - Xét tất cả những đỉnh v kề với u mà chưa được đánh dấu, với mỗi đỉnh v đó:
    - Đánh dấu v.
    - Ghi nhận vết đường đi từ u tới v (Có thể làm chung với việc đánh dấu)
    - Đẩy v vào hàng đợi (v sẽ chờ được duyệt tại những bước sau)
- Bước 3: Truy vết đường đi

BSF algorithm- Mã giả

```
Algorithm BFS(s)
Input: s is the source vertex
Output: Mark all vertices that can be visited from s

1. for each vertex v in graph do
2.     visited[v] := false;
3. Q = create empty queue;// Tại sao lại sử dụng QUEUE?
4. visited[s] := true;
5. enqueue(Q, s);
6. while Q is not empty do
7.     v := dequeue (Q);
8.     for k adjacent (kề) to v do
9.         if visited[k]=false then
10.            visited[k] := true;
11.            enqueue(Q, k);
```

Time Complexity of BFS-Using adjacency List

- Assume adjacency list
  - n = number of vertices, m = number of edges

```
1. for each vertex v in graph do
2.     visited[v] := false;
3. Q = create empty queue;
4. visited[s] := true;
5. enqueue(Q, s);
6. while Q is not empty do
7.     v := dequeue (Q);
8.     for k adjacent (kề) to v do
9.         if visited[k]=false then
10.            visited[k] := true;
11.            enqueue(Q, k);
```

O(n+m)

Each vertex will enter Q at most once

Each iteration takes time proportional to deg(v) + 1 (the number 1 is to include the case where deg(v) = 0)

Time Complexity of BFS-Using adjacency list

Running time:

- Given a graph with m edges, what is the total degree?

$$\sum_{vertex\ v} \deg(v) = 2m$$

- The **total** running time of the while loop is:

$$O(\sum_{vertex\ v} (\deg(v) + 1)) = O(n + m)$$

this is summing over all the iterations in the while loop!

Time Complexity of BFS-Using adjacency Matrix

- Assume adjacency matrix
  - n = number of vertices, m = number of edges

```
1. for each vertex v in graph do
2.     visited[v] := false;
3. Q = create empty queue;
4. visited[s] := true;
5. enqueue(Q, s);
6. while Q is not empty do
7.     v := dequeue (Q);
8.     for k adjacent (kề) to v do
9.         if visited[k]=false then
10.            visited[k] := true;
11.            enqueue(Q, k);
```

$O(n^2)$

Finding the adjacent vertices of v requires checking all elements in the row. This takes linear time O(n).

Summing over all the n iterations, the total running time is $O(n^2)$.

So, with adjacency matrix, BFS is $O(n^2)$ independent of number of edges m. With adjacent lists, BFS is O(n+m); if **$m=O(n^2)$** like a dense graph, **$O(n+m)=O(n^2)$**

Shortest Path Recording

- BFS we saw only tells us whether a path exists from source s, to other vertices v.
  - It doesn't tell us the path!
  - We need to modify the algorithm to record the path.

- How can we do that?
  - Note: we do not know which vertices lie on this path until we reach v!
  - Efficient solution:
    - Use an additional array trace[0..n-1]
    - trace[k] = v means that vertex k was visited from v

BSF algorithm using queue

- Đầu vào: đỉnh nguồn s
- Đầu ra: đánh dấu tất cả các đỉnh mà s có thể đi qua

- Bước 1: Khởi tạo:
  - Các đỉnh đều ở trạng thái chưa đánh dấu, ngoại trừ đỉnh xuất phát s là đã đánh dấu.
  - Một hàng đợi (Queue), ban đầu chỉ có một phần tử là s. Hàng đợi dùng để chứa các đỉnh sẽ được duyệt theo thứ tự ưu tiên chiều rộng
- Bước 2: Lặp các bước sau đến khi hàng đợi rỗng:
  - Lấy u khỏi hàng đợi, thông báo thăm u (Bắt đầu việc duyệt đỉnh u)
  - Xét tất cả những đỉnh v kề với u mà chưa được đánh dấu, với mỗi đỉnh v đó:
    - Đánh dấu v.
    - Ghi nhận vết đường đi từ u tới v (Có thể làm chung với việc đánh dấu)
    - Đẩy v vào hàng đợi (v sẽ chờ được duyệt tại những bước sau)
- Bước 3: Truy vết đường đi

BFS + Path Finding: Lưu vết tìm kiếm

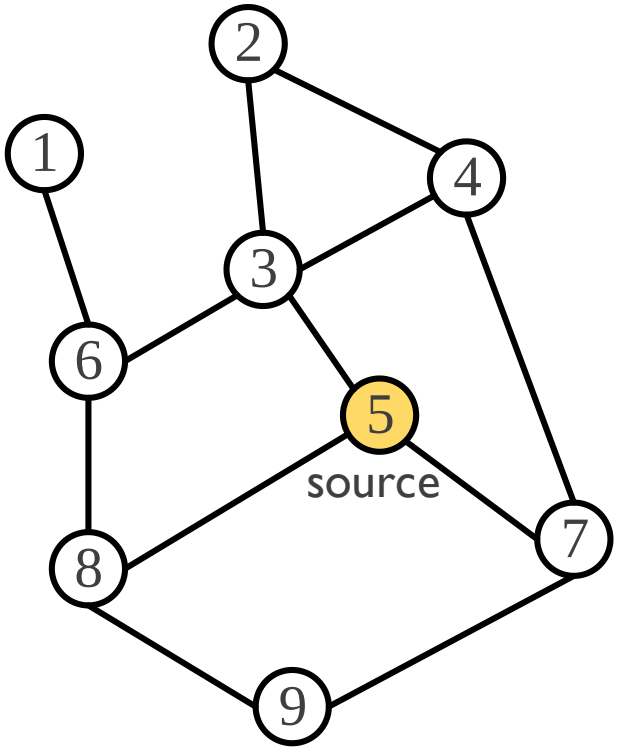
Algorithm BFS- Path Finding(s)

Input: s is the source vertex

Output: mảng trace lưu tất cả các đường dẫn ngắn nhất từ đỉnh s đến các đỉnh còn lại.

```
1. for each vertex v in graph do
2.     visited[v] := false;
3.     trace[v]  := -1;          ← initialize all trace[v] to -1
4. Q = create empty queue;
5. visited[s] := true;
6. enqueue(Q, s);
7. while Q is not empty do
8.     v := dequeue (Q);
9.     for k adjacent (kề) to v do
10.        if visited[k]=false then
11.            visited[k] := true;
12.            trace[k]   := v;      ← Record where you came from
13.            enqueue(Q, k);
14.        end if;
```

Example 1



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

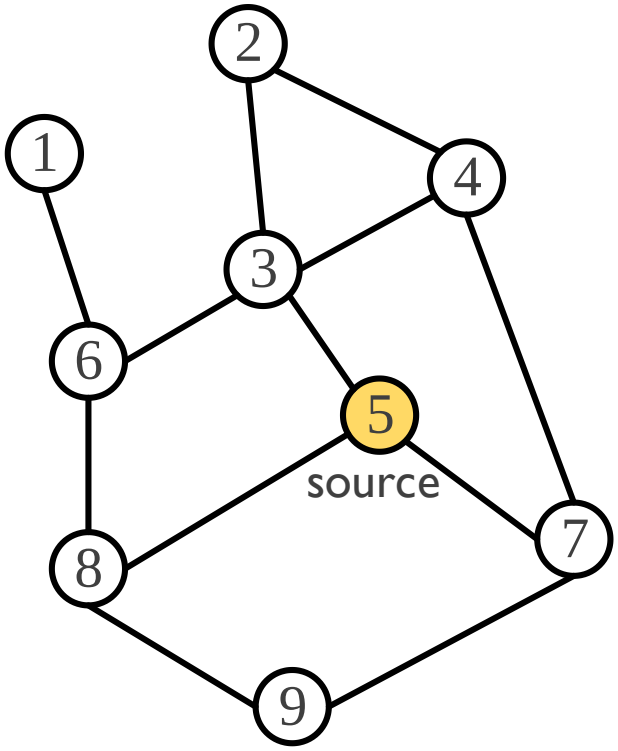
Visited & trace

1	False	-1
2	False	-1
3	False	-1
4	False	-1
5	False	-1
6	False	-1
7	False	-1
8	False	-1
9	False	-1

Initialize visited table (all False)

STT	Queue Q
B1	Empty

Example 1



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

Visited & trace

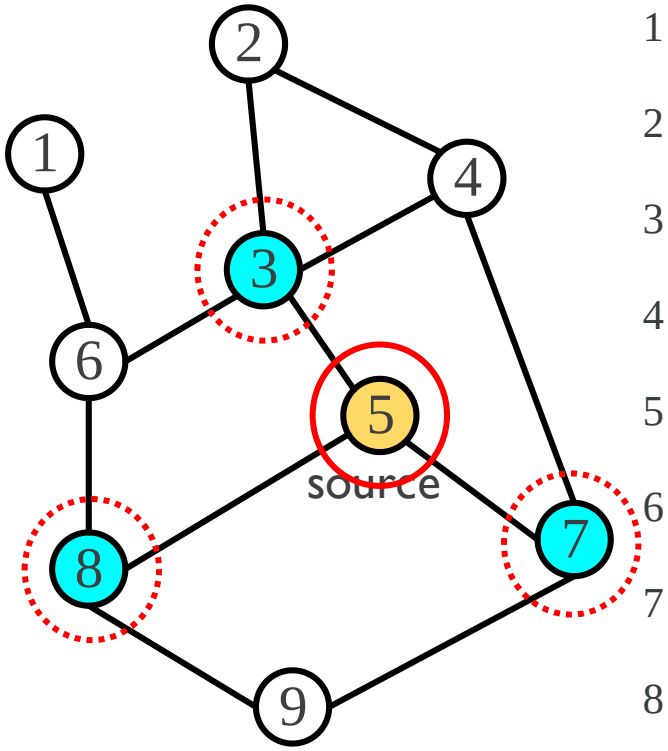
1	False	-1
2	False	-1
3	False	-1
4	False	-1
5	True	-1
6	False	-1
7	False	-1
8	False	-1
9	False	-1

STT	Queue Q
B1	Empty
B2	5

enqueue source 5 to Q

Mark 5 as visited.

Example 1



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

Visited & trace

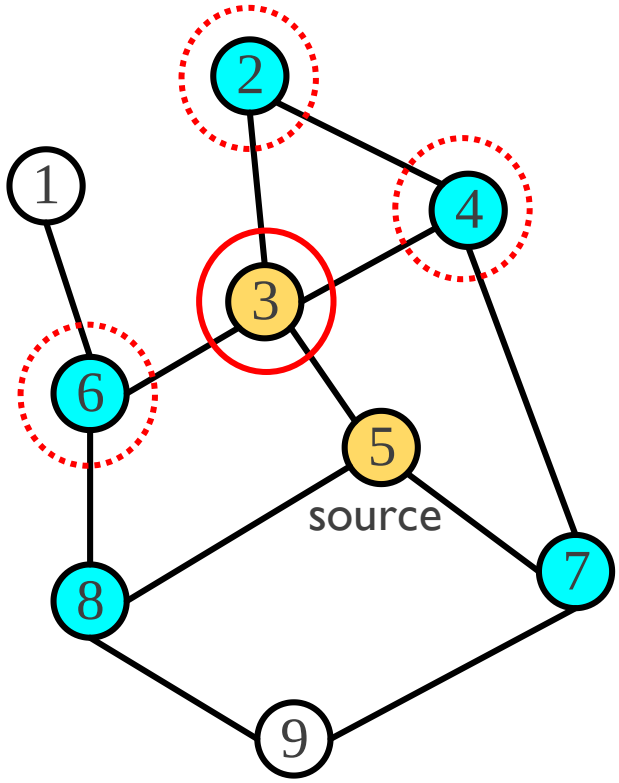
1	False	-1
2	False	-1
3	True	5
4	False	-1
5	True	-1
6	False	-1
7	True	5
8	True	5
9	False	-1

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8

Mark 3, 7, 8 as visited.

dequeue 5 on Q
enqueue **unvisited neighbors** of 5 ={3, 7, 8} on the queue

Example 1



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

Visited & trace

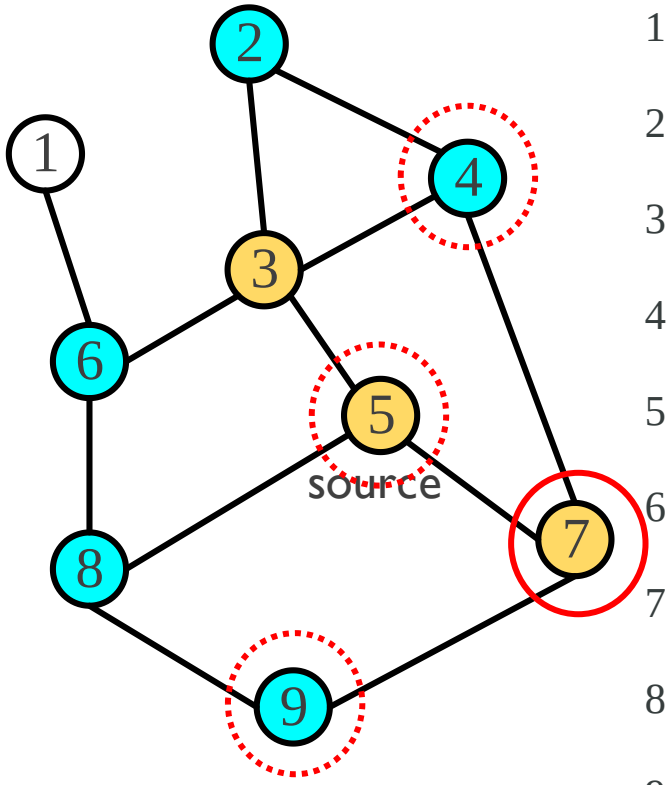
1	False	-1
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	False	-1

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6

Mark 2, 4, 6 as visited.

dequeue 3 on Q
enqueue **unvisited neighbors** of 3 ={2, 4, 6} on the queue
note: neighbors of 3 ={2, 4, 5, 6}, but 5 has been visited.

Example I



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

Visited & trace

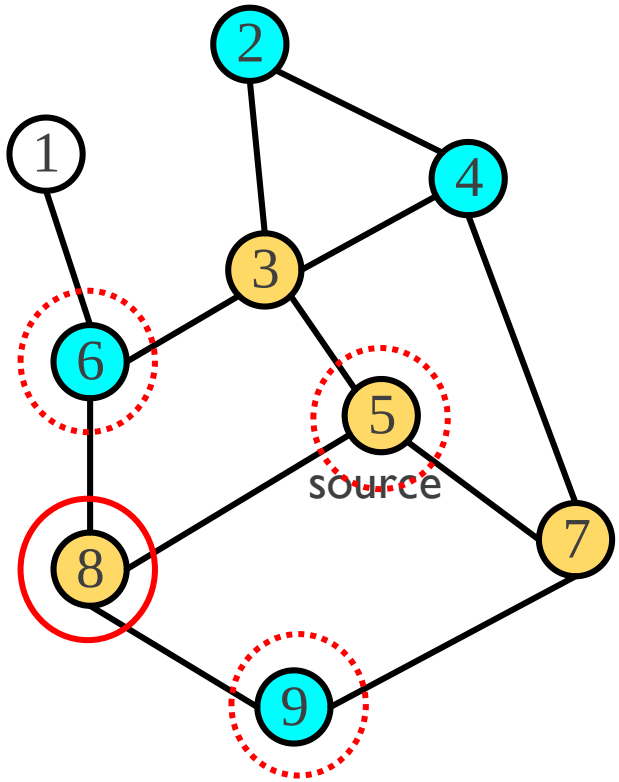
1	False	-1
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	True	7

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6
B5	8 2 4 6 9

Mark 9 as visited.

dequeue 7 on Q
enqueue **unvisited neighbors** of 7 ={9} on the queue
note: neighbors of 7 ={4, 5, 9}, but 4, 5 have been visited.

Example I



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

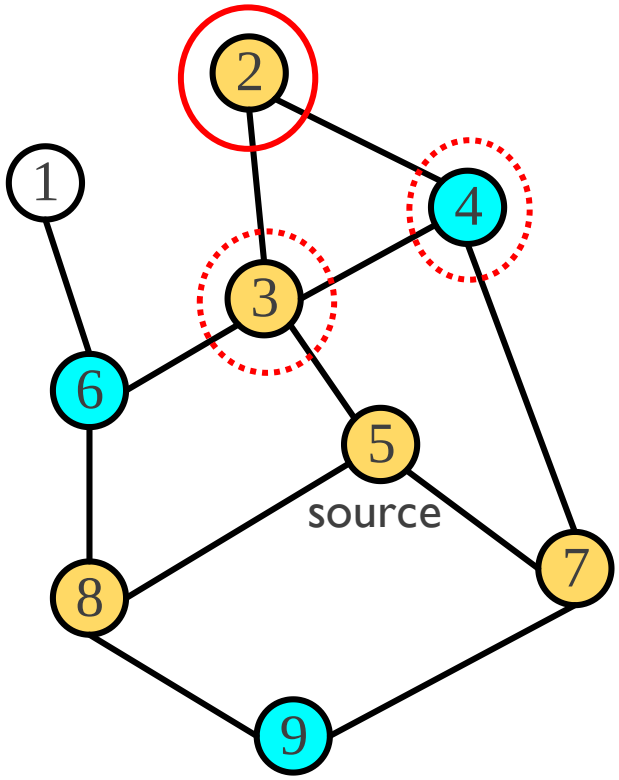
Visited & trace

1	False	-1
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	True	7

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6
B5	8 2 4 6 9
B6	2 4 6 9

dequeue 8 on Q
enqueue **unvisited neighbors** of 8 ={} on the queue
note: neighbors of 8 ={5, 6, 9}, but all have been visited.

Example I



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

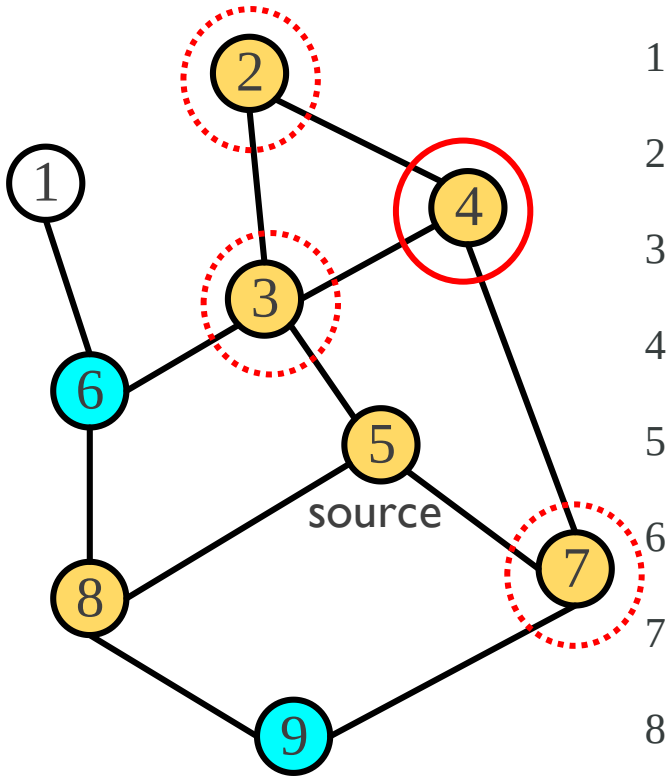
Visited & trace

1	False	-1
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	True	7

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6
B5	8 2 4 6 9
B6	2 4 6 9
B7	4 6 9

dequeue 2 on Q
enqueue **unvisited neighbors** of 2 ={} on the queue
note: neighbors of 2 ={3, 4}, but all have been visited.

Example I



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

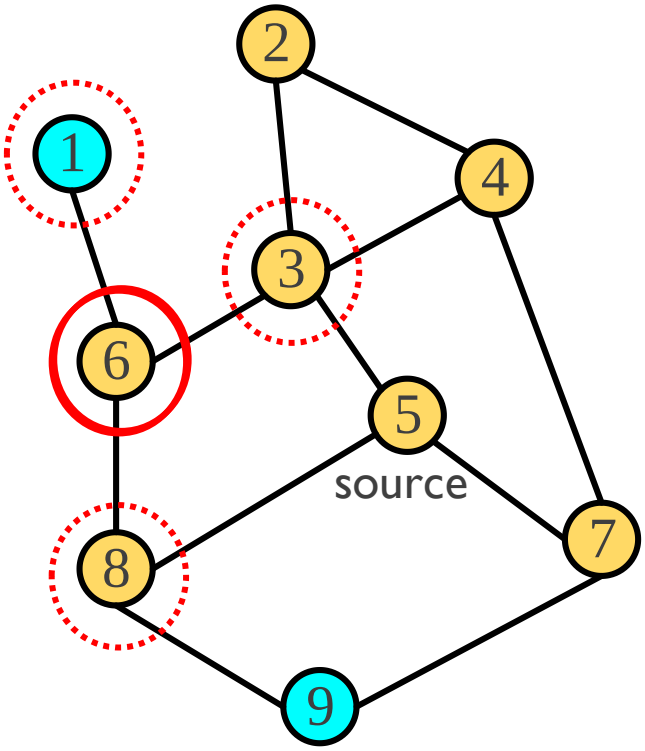
Visited & trace

1	False	-1
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	True	7

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6
B5	8 2 4 6 9
B6	2 4 6 9
B7	4 6 9
B8	6 9

dequeue 4 on Q
enqueue **unvisited neighbors** of 4 ={} on the queue
note: neighbors of 4 ={2, 3, 7}, but all have been visited.

Example 1



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

Visited & trace

1	True	6
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	True	7

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6
B5	8 2 4 6 9
B6	2 4 6 9
B7	4 6 9
B8	6 9
B9	9 1

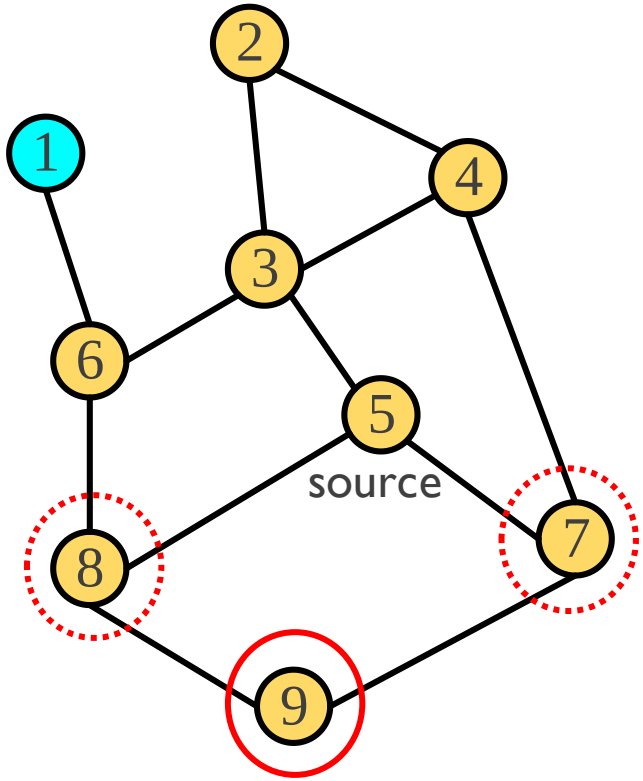
Mark 1 as visited.

dequeue 6 on Q

enqueue **unvisited neighbors** of 6 ={1} on the queue

note: neighbors of 6 ={1, 3, 8}, but 3, 8 have been visited.

Example I



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

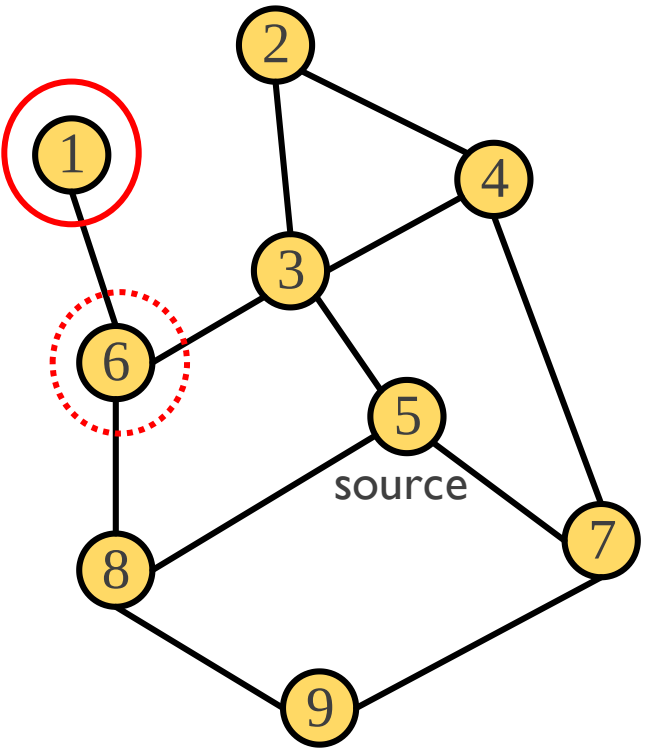
Visited & trace

1	True	6
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	True	7

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6
B5	8 2 4 6 9
B6	2 4 6 9
B7	4 6 9
B8	6 9
B9	9 1
B10	1

dequeue 9 on Q
enqueue **unvisited neighbors** of 9 ={} on the queue
note: neighbors of 9 ={7, 8}, but all have been visited.

Example 1



Adjacency List

1	6
2	3 4
3	2 4 5 6
4	2 3 7
5	3 7 8
6	1 3 8
7	4 5 9
8	5 6 9
9	7 8

Visited & trace

1	True	6
2	True	3
3	True	5
4	True	3
5	True	-1
6	True	3
7	True	5
8	True	5
9	True	7

STT	Queue Q
B1	Empty
B2	5
B3	3 7 8
B4	7 8 2 4 6
B5	8 2 4 6 9
B6	2 4 6 9
B7	4 6 9
B8	6 9
B9	9 1
B10	1
B11	Empty

dequeue 1 on Q
enqueue unvisited neighbors of 1 ={} on the queue
note: neighbors of 1 ={6}, but 6 has been visited. => Q is Empty

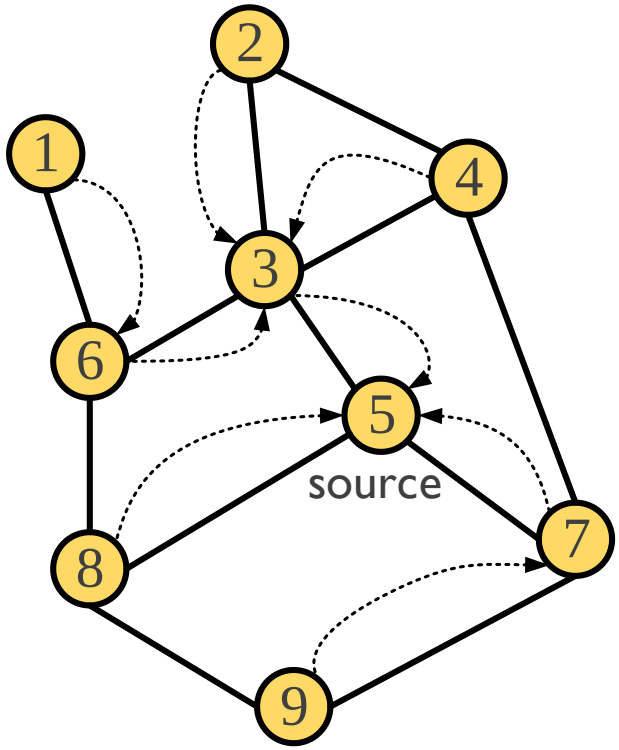
STOP

What did we discover?
Look at "visited" array. There exists a path from source vertex 5 to all vertices in the graph

Path Finding – Truy vết đường đi

```
Algorithm PATH(v)
input: vertex v
output: the shortest path
from s to v (minimum
number of edges).

1. if trace[v] <> -1 then
2.         PATH(trace[v]);
3. print v;
```



trace	
1	6
2	3
3	5
4	3
5	-1
6	3
7	5
8	5
9	7

Try some examples, report path from s to v:

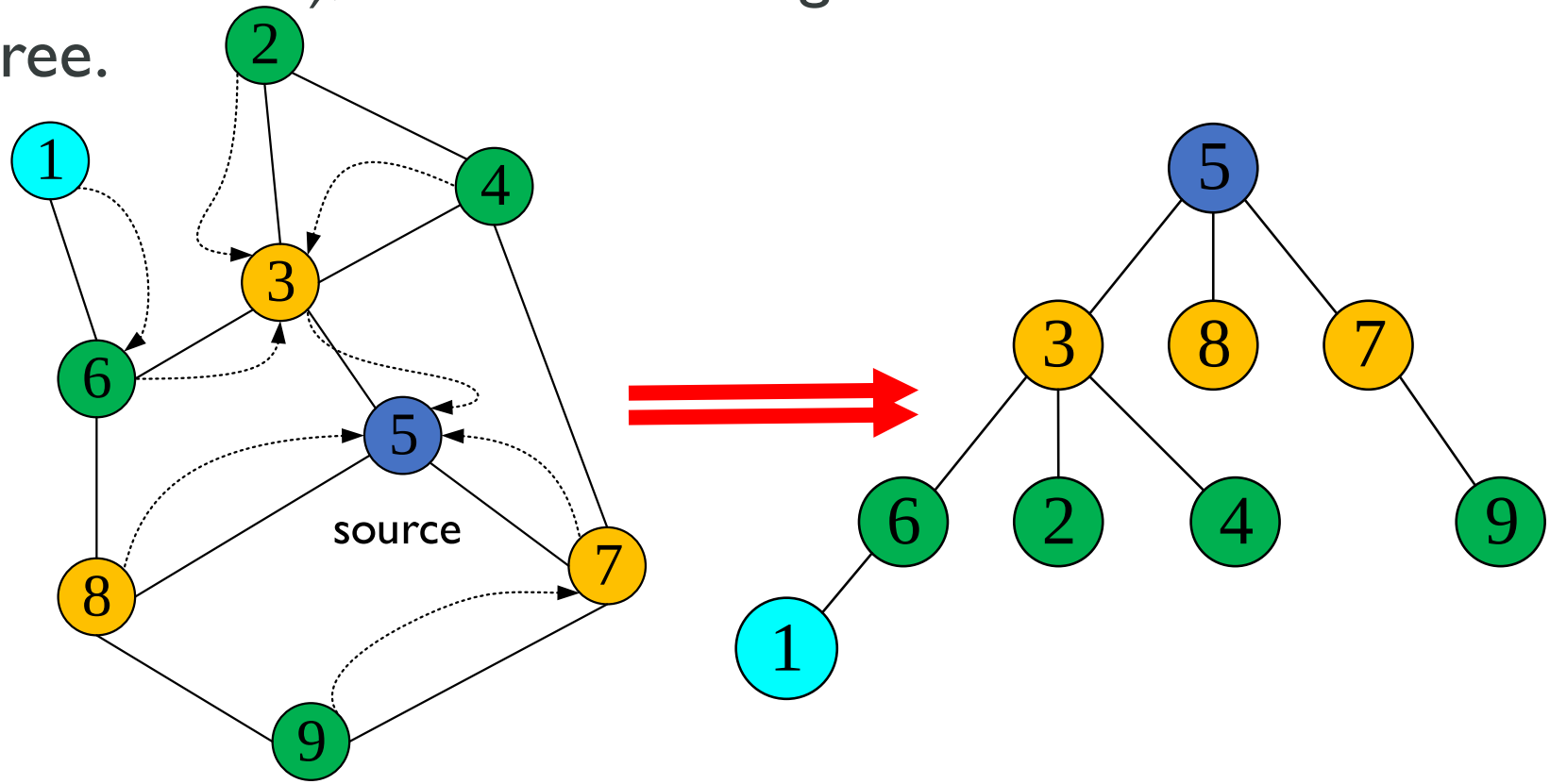
PATH(1) → ?

PATH(2) → ?

PATH(9)→ ?

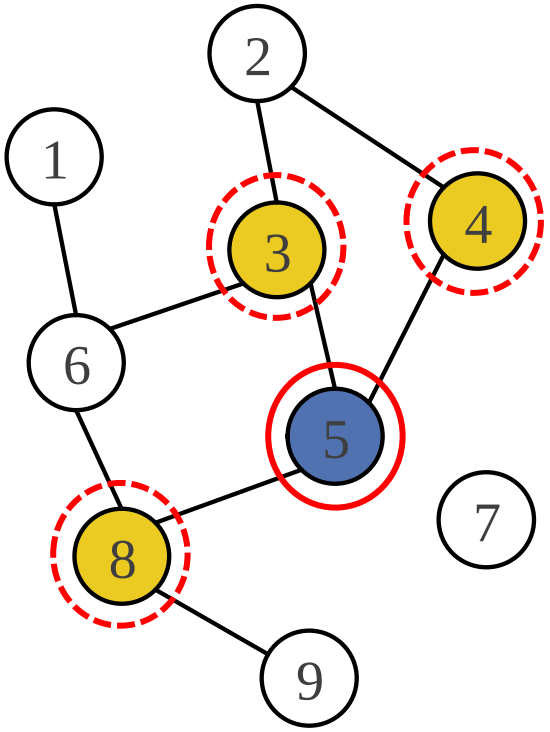
BFS tree

- The paths found by BFS is often drawn as a rooted tree (called BFS tree), with the starting vertex as the root of the tree.



- Question: What would a "level" order traversal tell you?

Example 2

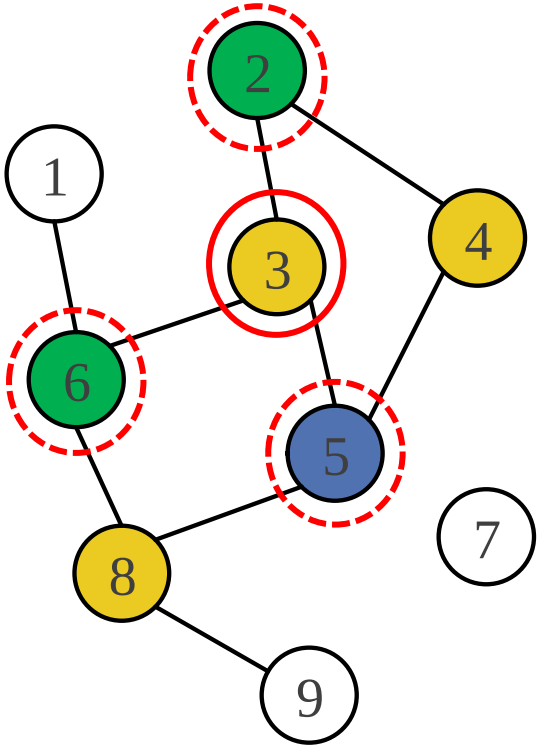


STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8

Visited & Trace

u	visited	trace
1	F	-1
2	F	-1
3	T	5
4	T	5
5	T	-1
6	F	-1
7	F	-1
8	T	5
9	F	-1

Example 2

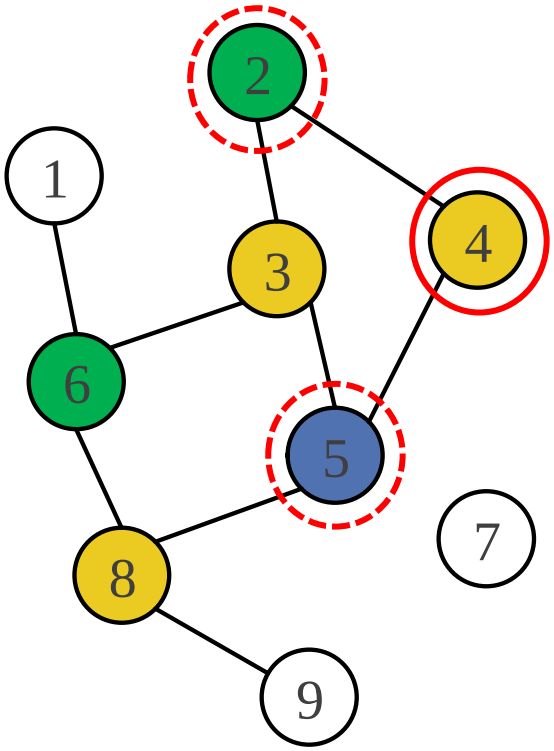


STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8
S2	3 4 8	3	2 6	4 8 2 6

Visited & Trace

u	visited	trace
1	F	-1
2	T	3
3	T	5
4	T	5
5	T	-1
6	T	3
7	F	-1
8	T	5
9	F	-1

Example 2

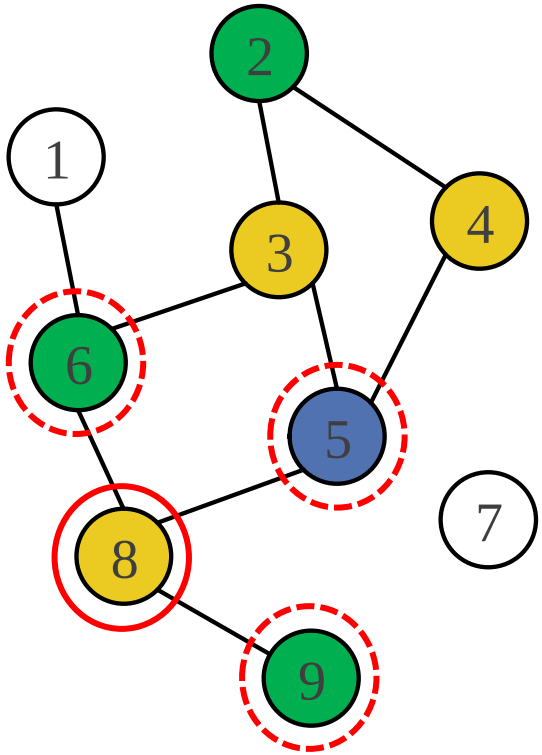


STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8
S2	3 4 8	3	2 6	4 8 2 6
S3	4 8 2 6	4	Ø	8 2 6

Visited & Trace

u	visited	trace
1	F	-1
2	T	3
3	T	5
4	T	5
5	T	-1
6	T	3
7	F	-1
8	T	5
9	F	-1

Example 2

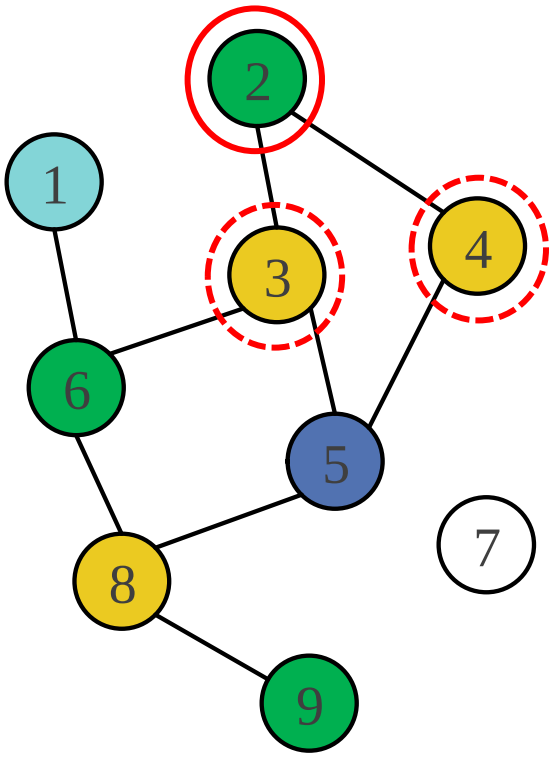


STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8
S2	3 4 8	3	2 6	4 8 2 6
S3	4 8 2 6	4	Ø	8 2 6
S4	8 2 6	8	9	2 6 9

Visited & Trace

u	visited	trace
1	F	-1
2	T	3
3	T	5
4	T	5
5	T	-1
6	T	3
7	F	-1
8	T	5
9	T	8

Example 2

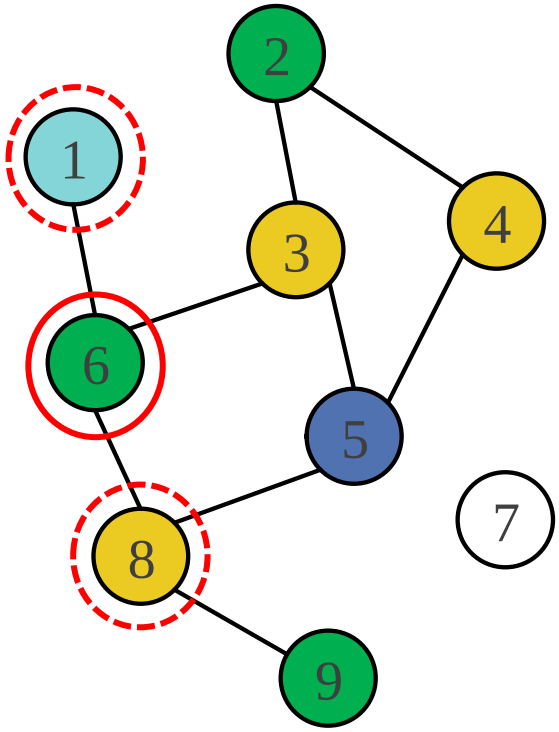


STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8
S2	3 4 8	3	2 6	4 8 2 6
S3	4 8 2 6	4	Ø	8 2 6
S4	8 2 6	8	9	2 6 9
S5	2 6 9	2	Ø	6 9

Visited & Trace

u	visited	trace
1	F	-1
2	T	3
3	T	5
4	T	5
5	T	-1
6	T	3
7	F	-1
8	T	5
9	T	8

Example 2

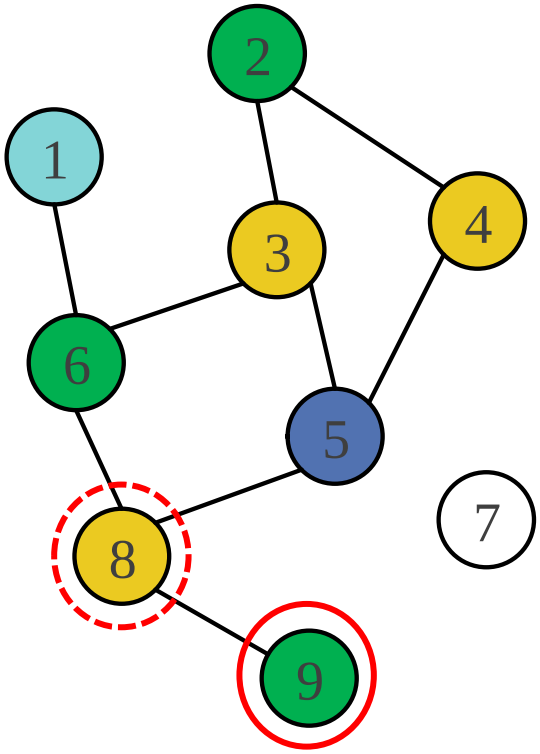


STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8
S2	3 4 8	3	2 6	4 8 2 6
S3	4 8 2 6	4	∅	8 2 6
S4	8 2 6	8	9	2 6 9
S5	2 6 9	2	∅	6 9
S6	6 9	6	1	9 1

Visited & Trace

u	visited	trace
1	T	6
2	T	3
3	T	5
4	T	5
5	T	-1
6	T	3
7	F	-1
8	T	5
9	T	8

Example 2

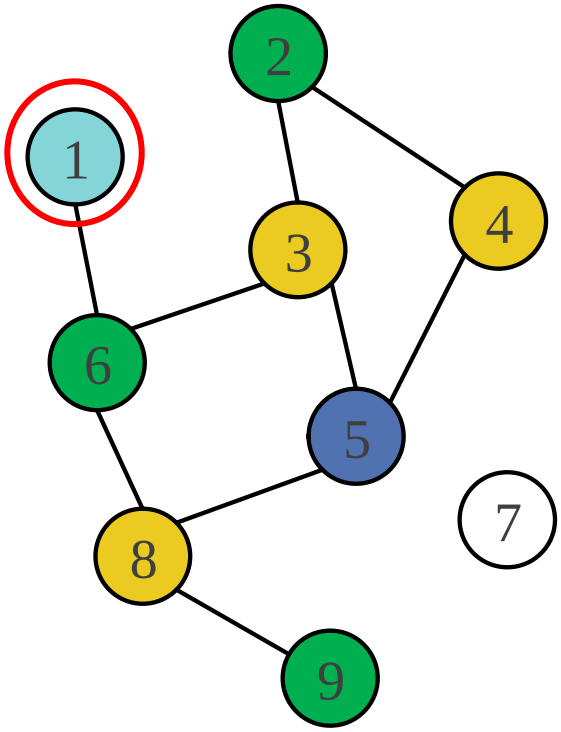


STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8
S2	3 4 8	3	2 6	4 8 2 6
S3	4 8 2 6	4	Ø	8 2 6
S4	8 2 6	8	9	2 6 9
S5	2 6 9	2	Ø	6 9
S6	6 9	6	1	9 1
S7	9 1	9	Ø	1

Visited & Trace

u	visited	trace
1	T	6
2	T	3
3	T	5
4	T	5
5	T	-1
6	T	3
7	F	-1
8	T	5
9	T	8

Example 2



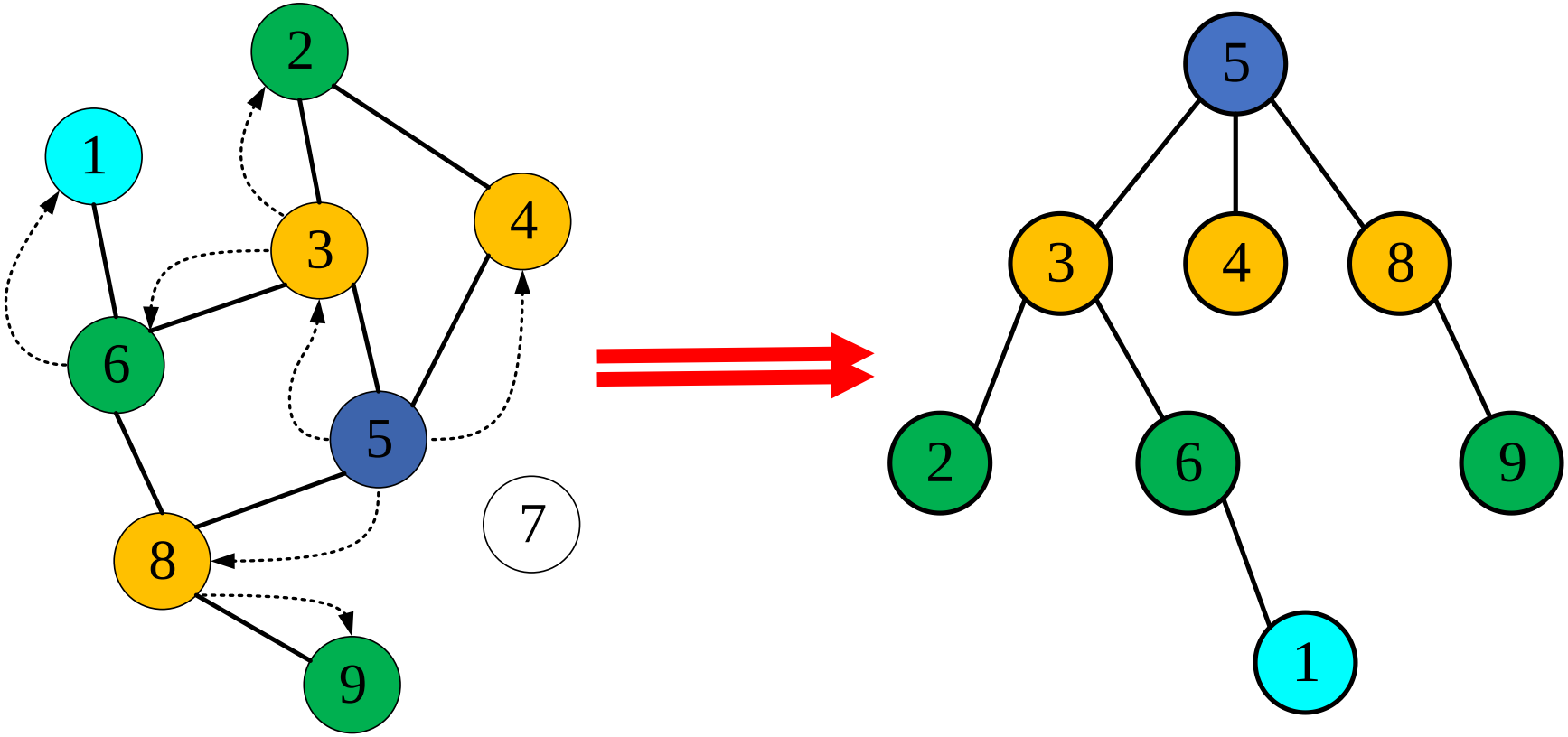
STEP	QUEUE Q	v= dequeue (Q)	All k adjacent to v (and unvisited)	QUEUE Q AFTER STEP
S1	5	5	3 4 8	3 4 8
S2	3 4 8	3	2 6	4 8 2 6
S3	4 8 2 6	4	Ø	8 2 6
S4	8 2 6	8	9	2 6 9
S5	2 6 9	2	Ø	6 9
S6	6 9	6	1	9 1
S7	9 1	9	Ø	1
S8	1	1	Ø	Ø

Visited & Trace

u	visited	trace
1	T	6
2	T	3
3	T	5
4	T	5
5	T	-1
6	T	3
7	F	-1
8	T	5
9	T	8

Example 2

BFS-Tree



How do we record the shortest distances?

Algorithm BFS(s)

```
1. for each vertex v in graph do
2.     visited[v] := false;
3.     pred[v]   := -1; d(v)= ∞;
4. Q = create empty queue;
5. visited[s] := true; d(v)= 0;
6. enqueue(Q, s);
7. while Q is not empty do
8.     v := dequeue (Q);
9.     for k adjacent (kề) to v do
10.        if visited[k]=false then
11. d(v)= d(v)+1; visited[k] := true;
12.               pred[k] := v;
13.               enqueue(Q, k);
```

Application of BFS

- One application concerns how to find connected components in a graph

- If a graph has more than one connected components, BFS builds a BFS-forest (not just BFS-tree)!
  - Each tree in the forest is a **connected component**.

Nhận xét: TODO

- Ưu điểm
  - Xét duyệt tất cả các đỉnh để trả về kết quả.
  - Nếu số đỉnh là hữu hạn, thuật toán chắc chắn tìm ra kết quả.

- Khuyết điểm
  - Mang tính chất vét cạn, không nên áp dụng nếu duyệt số đỉnh quá lớn.
  - Mang tính chất mù quáng, duyệt tất cả đỉnh, không chú ý đến thông tin trong các đỉnh để duyệt hiệu quả, dẫn đến duyệt qua các đỉnh không cần thiết.

NỘI DUNG

- Đồ thị và các khái niệm trên đồ thị
- Biểu diễn đồ thị trên máy tính
- Duyệt đồ thị theo chiều sâu (DFS)
- Duyệt đồ thị theo chiều rộng (BFS)
- Một số ứng dụng
- **Bài tập**

BÀI TẬP

Bài 1: Tìm hiểu và cài đặt hàm biểu diễn đồ thị bằng danh sách liên kết.

Bài 2: Viết hàm cài đặt xác định số thành phần liên thông của đồ thị và cho biết đồ thị có liên thông

Bài 3: Viết hàm cho biết đồ thị có chu trình không ? Nếu có liệt kê các chu trình của đồ thị?

Chúc các em học tốt!