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

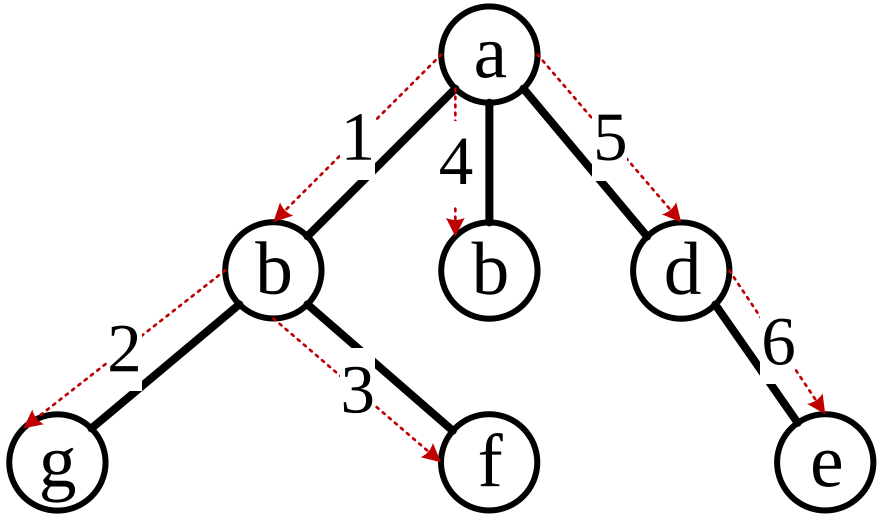
- Two common graph traversal algorithms
  - Duyệt theo chiều rộng sử dụng thuật toán Breadth-First Search (BFS)
    - Find the shortest paths in an unweighted graph
  - Duyệt theo chiều sâu sử dụng thuật toán Depth-First Search (DFS)

NỘI DUNG

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

Định nghĩa DFS

- Từ đỉnh (nút) gốc ban đầu, thuật toán duyệt đi xa nhất theo từng nhánh, khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo. Quá trình duyệt chỉ dừng lại khi tìm thấy đỉnh cần tìm hoặc tất cả đỉnh đều đã được duyệt qua.
- Ví dụ về thứ tự duyệt DFS trên cây như bên dưới:



Định nghĩa DFS

- DFS is another popular graph search strategy
  - Idea is similar to pre-order traversal (visit children first)

- DFS can provide certain information about the graph that BFS cannot
  - It can tell whether we have encountered a cycle or not

Thuật toán DFS

- Thuật toán duyệt theo chiều sâu (Depth-First Search)

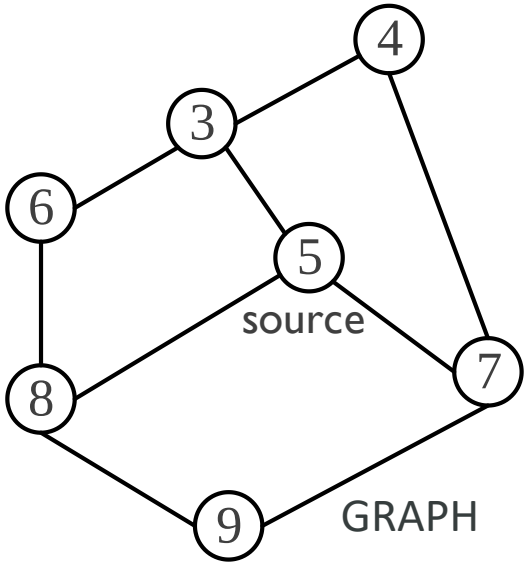
  - Cho $G = (V, E)$ là đồ thị có tập các đỉnh V và tập các cạnh E. v là một đỉnh trong V và u là đỉnh kề của v, sao cho u cũng thuộc V.
  - Khi đó ta dán nhãn cho tất cả các đỉnh của đồ thị là 0. Chọn một đỉnh v thuộc tập V để bắt đầu duyệt. Gán nhãn đỉnh v này là 1: v đã được duyệt.
  - Chọn đỉnh u trong tập V kề với đỉnh v mà nhãn là 0. Duyệt qua đỉnh u và gán nhãn u là 1. Tiếp tục quá trình duyệt đến khi tất cả các đỉnh đồ thị có nhãn là 1

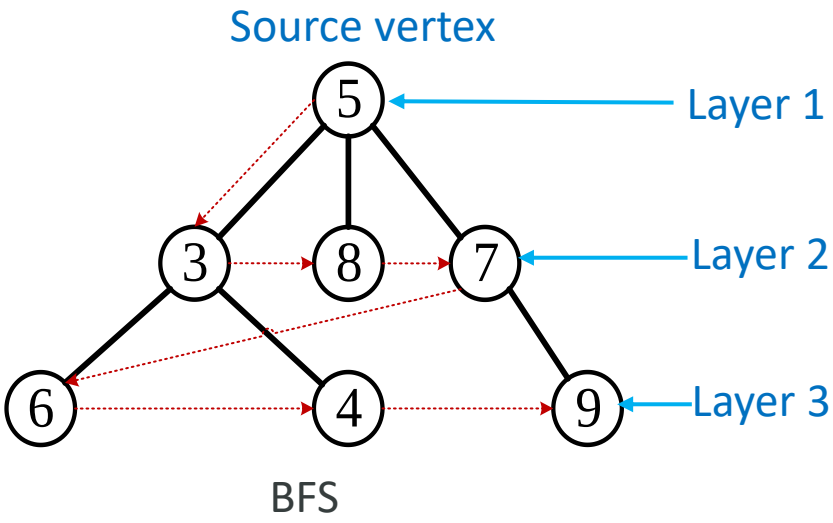
Định nghĩa DFS

- Từ đỉnh (nút) gốc ban đầu, thuật toán duyệt đi xa nhất theo từng nhánh, khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo. Quá trình duyệt chỉ dừng lại khi tìm thấy đỉnh cần tìm hoặc tất cả đỉnh đều đã được duyệt qua.

Duyệt đồ thị theo chiều rộng

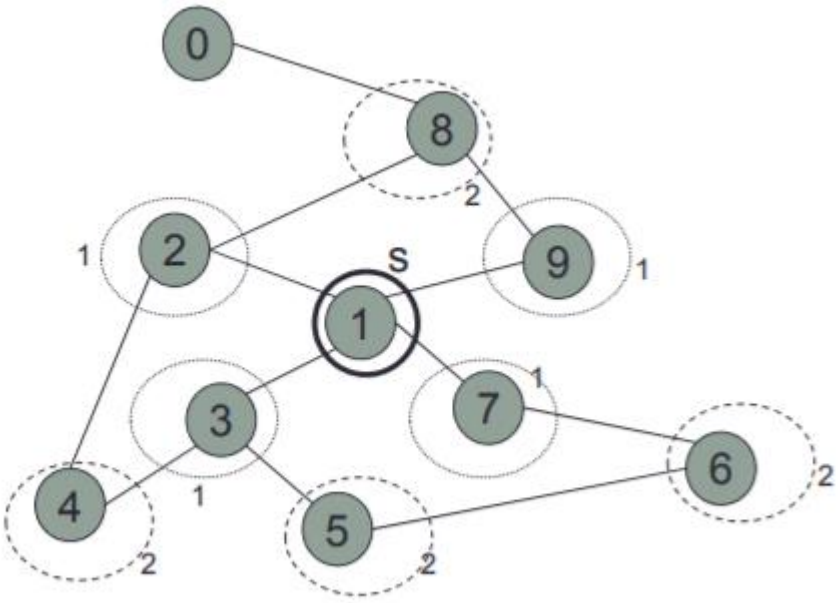
- BFS is a traversing algorithm where you should start traversing from a selected vertex (source or starting node) and traverse the graph layer-wise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.
- Như 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
- Xem ví dụ bên: Theo BFS, phải đi qua đỉnh {5} trong layer 1 trước khi bạn di chuyển đến các nút {3, 8, 7} trong layer 2, rồi đến các node {6, 4, 9} layer 3.





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

Thuật toán DFS

- Thuật toán duyệt theo chiều sâu (Depth-First Search)
  - Cho $G = (V, E)$ là đồ thị có tập các đỉnh V và tập các cạnh E. v là một đỉnh trong V và u là đỉnh kề của v, sao cho u cũng thuộc V.
  - Khi đó ta dán nhãn cho tất cả các đỉnh của đồ thị là **false**. Chọn một đỉnh v thuộc tập V để bắt đầu duyệt. Gán nhãn đỉnh v này là **true** (v đã được duyệt).
  - Chọn đỉnh u trong tập V kề với đỉnh v mà nhãn là **false**. Duyệt qua đỉnh u và gán nhãn u là **true**. Tiếp tục quá trình duyệt đến khi tất cả các đỉnh đồ thị có nhãn là **true**.

Cài đặt BSF

Để cài đặt thuật toán DSF, ta có các cách sau:

1. Sử dụng đệ quy
2. Sử dụng stack
3. …

DSF algorithm – using stack

```
Algorithm DFS(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;          ← mark all vertices false
3.     trace[v]   := -1;             ← initialize all trace[v] to -1
4. S = create empty stack;
5. visited[s] := true;
6. push(S, s);
7. while S is not empty do
8.     v := top(S);
9.     pop(S);
9.     if v exists the unvisited adjacent vertex then
10.        Choose any 1 node k in the unvisited adjacent vertices of v then
11.            visited[k] := true;
12.            trace[k] := v;        ← Record where you came from
13.            push(S, v);           ← đưa v vào lại stack
14.            push(S, k);           ← đưa tiếp k vào stack
```

Chú ý: việc chọn 1 đỉnh kề để xét có thể chọn bất kỳ hoặc theo trật tự cho trước. Vì vậy đường đi từ đỉnh s tới đỉnh d có thể có nhiều đường.

DSF algorithm – using recursive

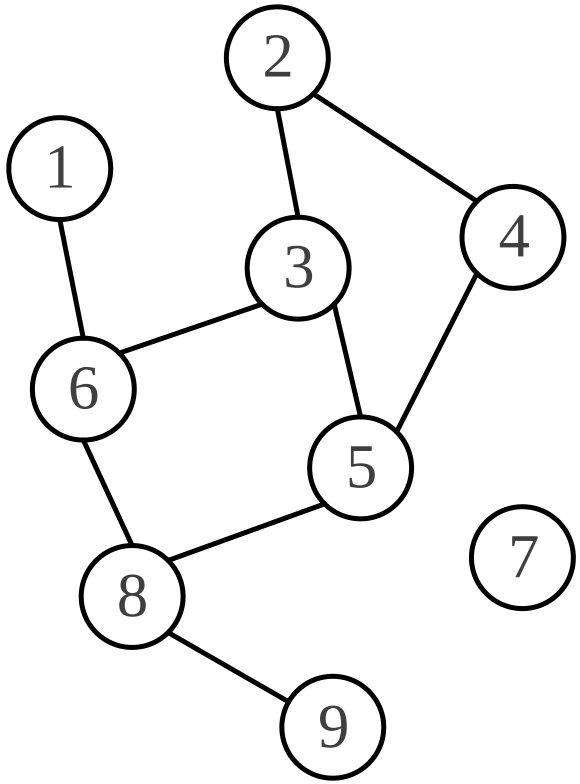
```
Algorithm DFS(s)
Input: s is the source vertex
Output: Mark all vertices that can be visited from s
1. for each vertex v in graph do
2.    visited[v] := false; // all vertices are not visited
3.    trace[v]   := -1;
4. RDFS(s);

Algorithm RDFS(v)
1. flag[v] := true; // vertex v is visited
2. for k adjacent to v do
3.   if visited[k] = false then
4.      trace[k] := v;
5.      RDFS(k); // unvisited neighbor call RDFS(k)
```

DSF algorithm

Chương trình chính:

Input: Nhập dữ liệu: đồ thị, đỉnh xuất phát s, đỉnh đích d
DFS(S);
if D is not marked then
    output: Không thể có đường đi từ s tới d
else Truy theo vết để tìm đường đi từ s tới d

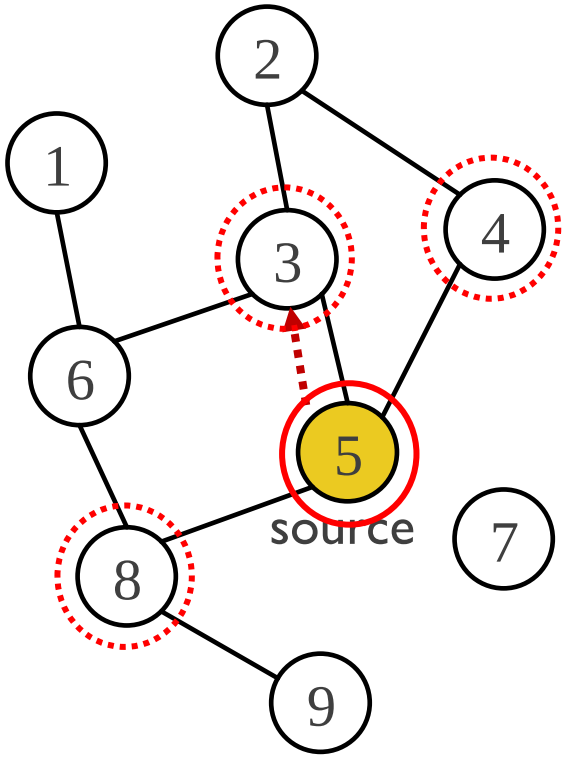
Example I



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

Example 1



STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3

Pop(S)

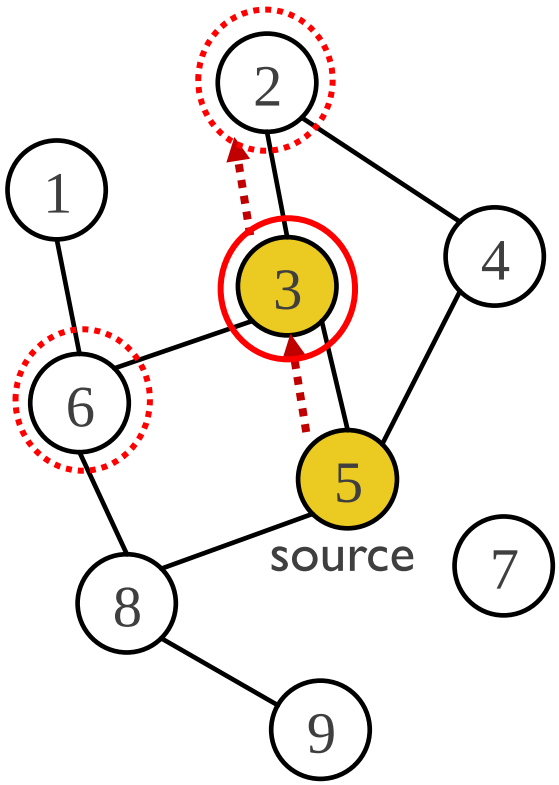
Chú ý: việc chọn 1 đỉnh kề để xét có thể chọn bất kỳ hoặc theo trật tự cho trước. → Vì vậy đường đi từ đỉnh s tới đỉnh d có thể có nhiều đường.

Ví dụ bên trên chọn đỉnh kề của 5 để xét là đỉnh 3 thay vì 8.

Visited & Trace

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

Example 1

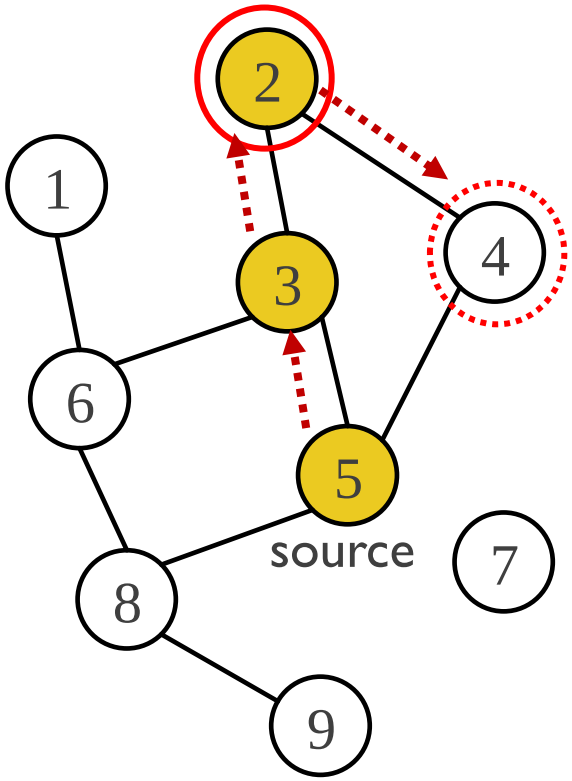


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 **3**	3	2	5 3 2

Visited & Trace

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

Example 1

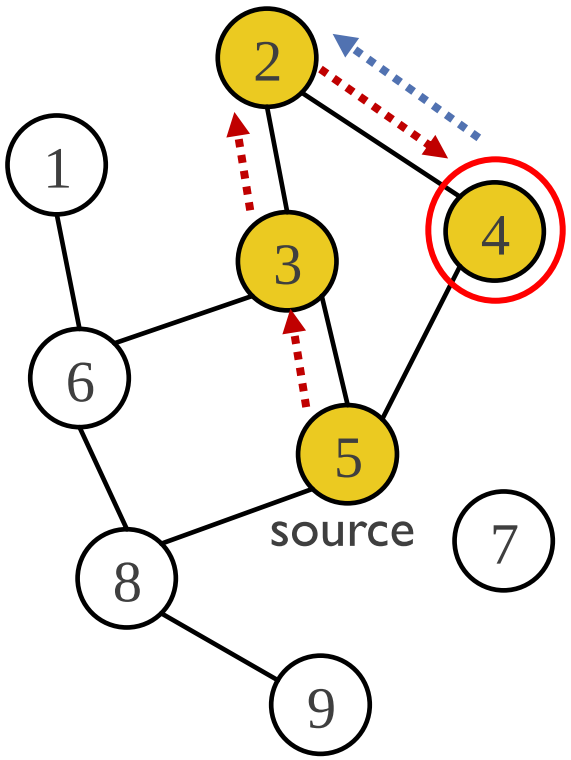


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4

Visited & Trace

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

Example 1

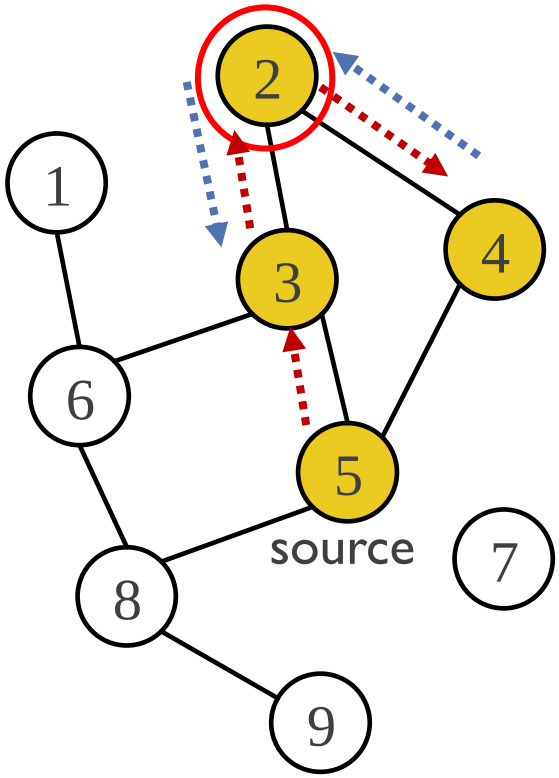


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2

Visited & Trace

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

Example 1

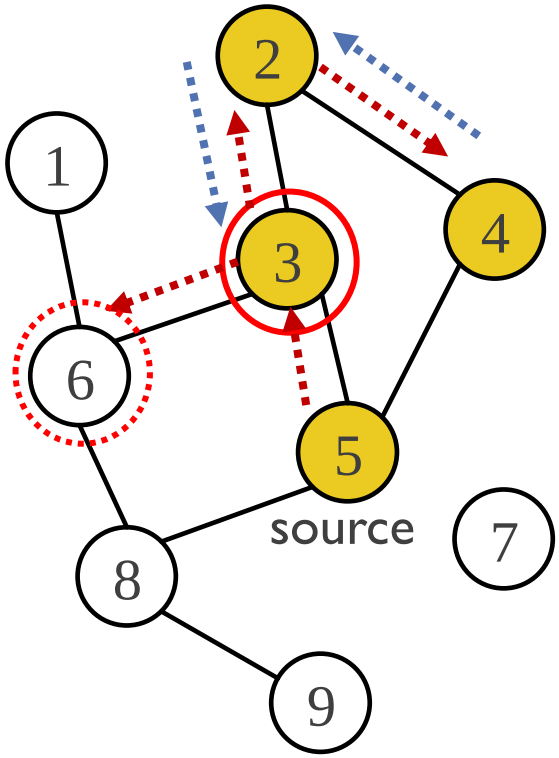


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2
5	5 3 2	2	Ø	5 3

Visited & Trace

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

Example 1

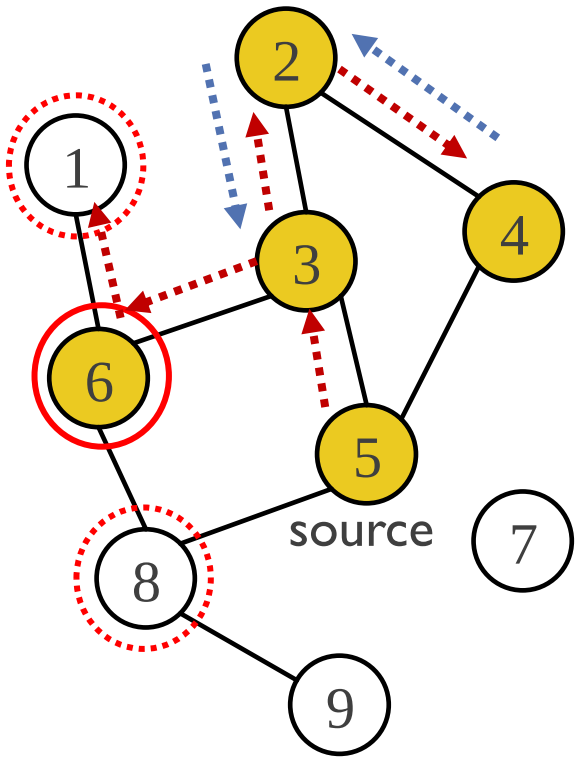


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	∅	5 3 2
5	5 3 2	2	∅	5 3
6	5 3	3	6	5 3 6

Visited & Trace

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

Example 1

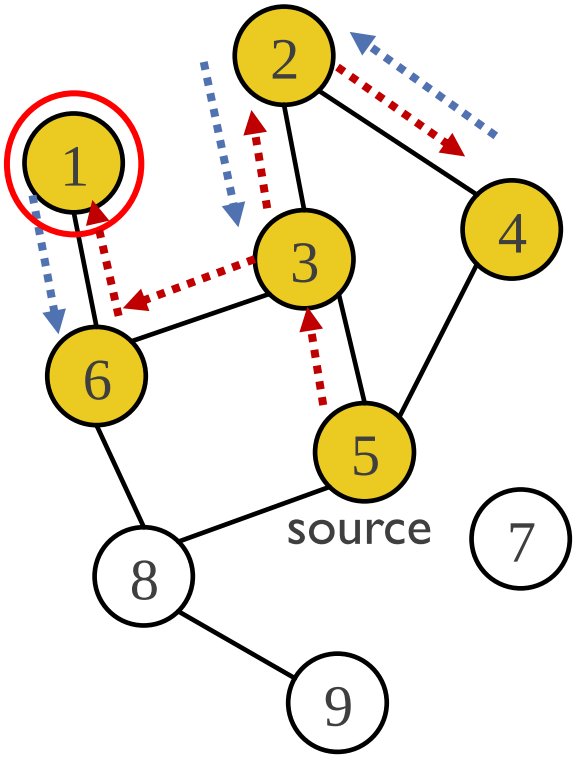


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	∅	5 3 2
5	5 3 2	2	∅	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1

Visited & Trace

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

Example 1

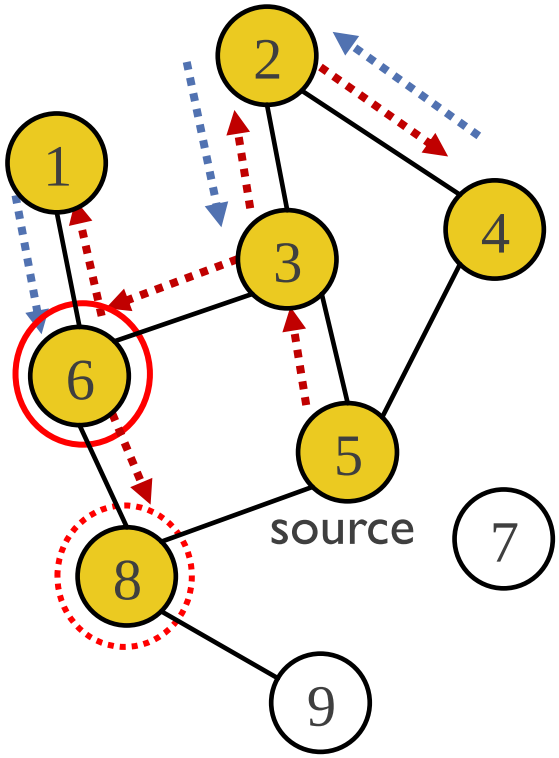


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2
5	5 3 2	2	Ø	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	Ø	5 3 6

Visited & Trace

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

Example 1

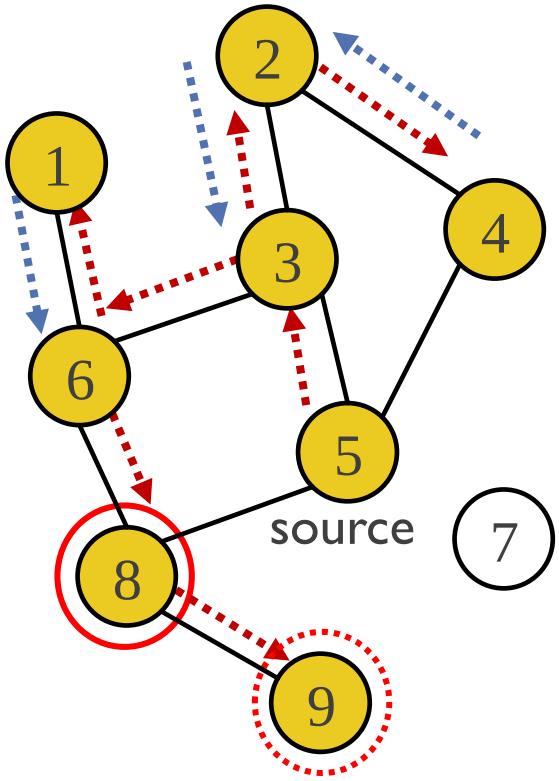


STEP	STACK	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	∅	5 3 2
5	5 3 2	2	∅	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	∅	5 3 6
9	5 3 6	6	8	5 3 6 8

Visited & Trace

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

Example 1

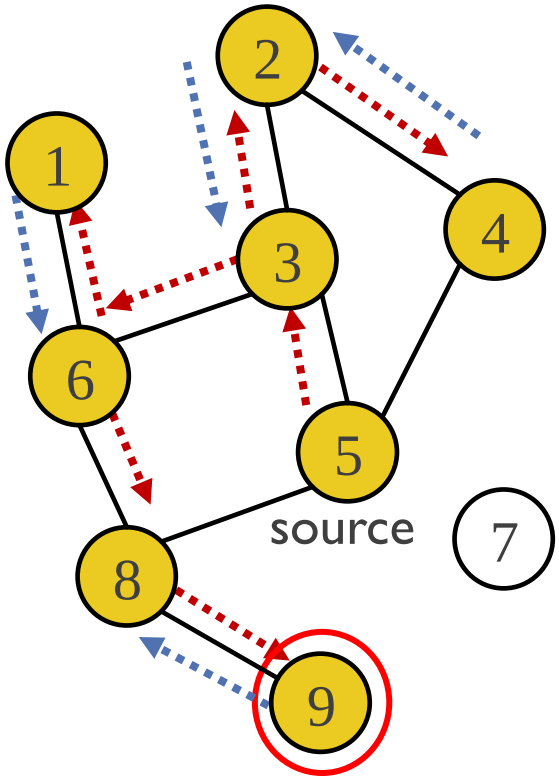


STEP	STACK	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	∅	5 3 2
5	5 3 2	2	∅	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	∅	5 3 6
9	5 3 6	6	8	5 3 6 8
10	5 3 6 8	8	9	5 3 6 8 9

Visited & Trace

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

Example 1

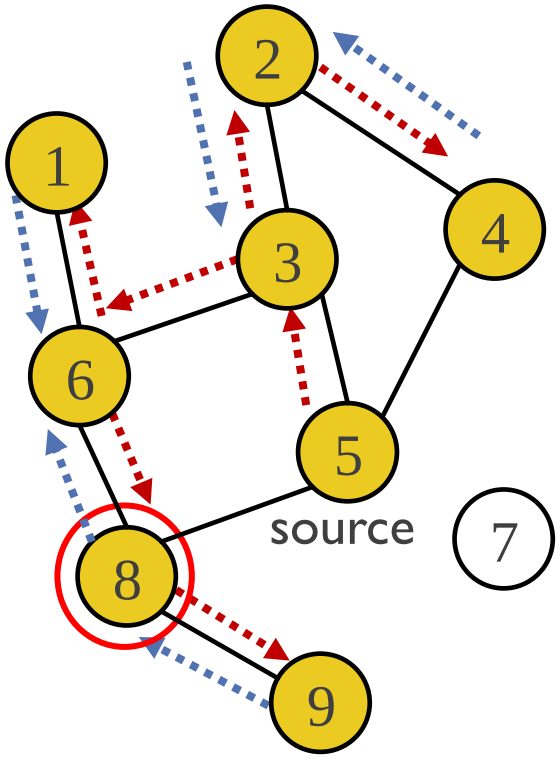


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2
5	5 3 2	2	Ø	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	Ø	5 3 6
1	5 3 6	6	8	5 3 6 8
10	5 3 6 8	8	9	5 3 6 8 9
11	5 3 6 8 9	9	Ø	5 3 6 8

Visited & Trace

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

Example 1

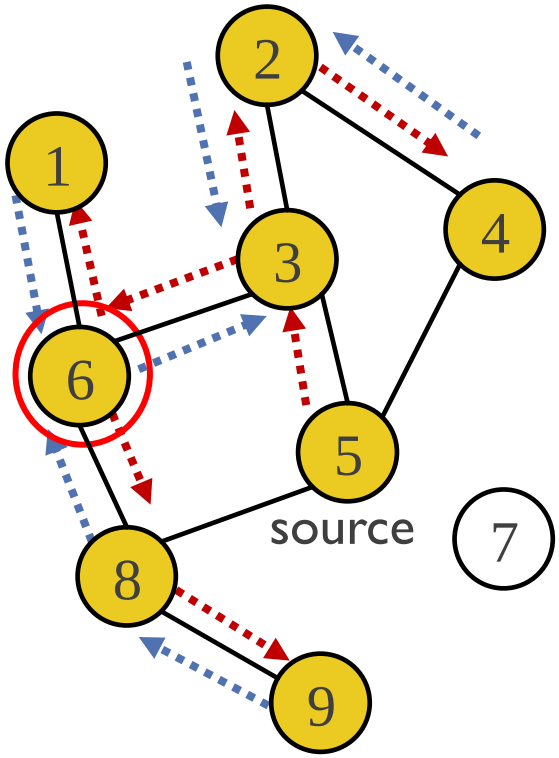


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2
5	5 3 2	2	Ø	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	Ø	5 3 6
9	5 3 6	6	8	5 3 6 8
10	5 3 6 8	8	9	5 3 6 8 9
11	5 3 6 8 9	9	Ø	5 3 6 8
12	5 3 6 8	8	Ø	5 3 6

Visited & Trace

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

Example 1

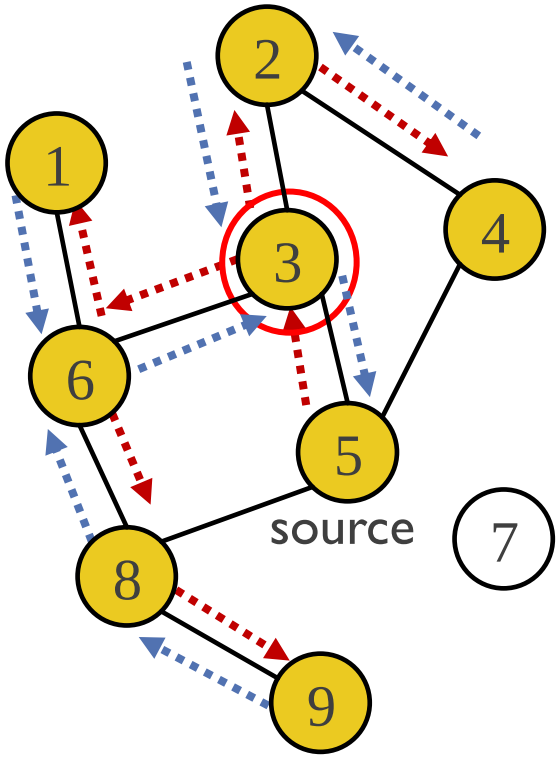


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2
5	5 3 2	2	Ø	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	Ø	5 3 6
9	5 3 6	6	8	5 3 6 8
10	5 3 6 8	8	9	5 3 6 8 9
11	5 3 6 8 9	9	Ø	5 3 6 8
12	5 3 6 8	8	Ø	5 3 6
13	5 3 6	6	Ø	5 3

Visited & Trace

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

Example 1

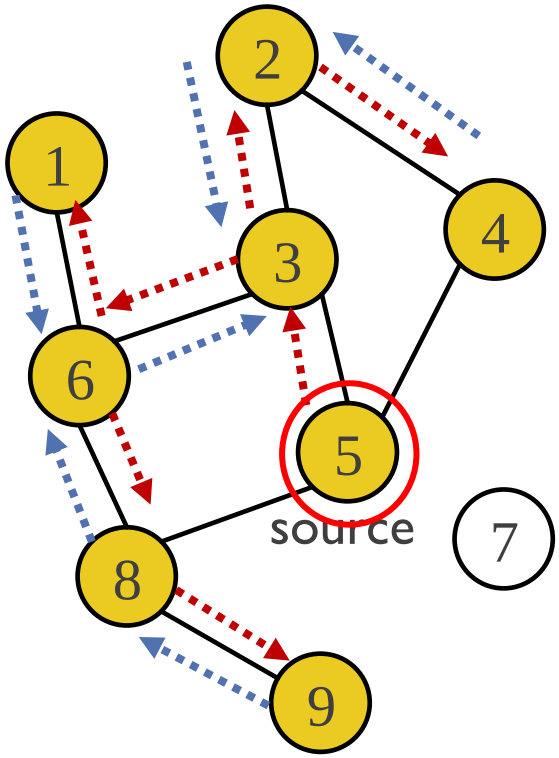


STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2
5	5 3 2	2	Ø	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	Ø	5 3 6
9	5 3 6	6	8	5 3 6 8
10	5 3 6 8	8	9	5 3 6 8 9
11	5 3 6 8 9	9	Ø	5 3 6 8
12	5 3 6 8	8	Ø	5 3 6
13	5 3 6	6	Ø	5 3
14	5 3	3	Ø	5

Visited & Trace

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

Example 1



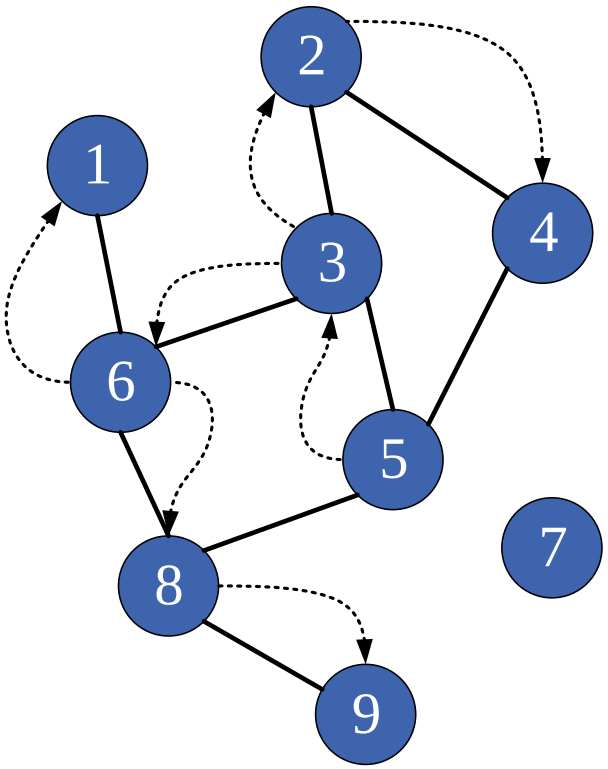
STEP	STACK S	u	v	STACK S AFTER STEP
1	5	5	3	5 3
2	5 3	3	2	5 3 2
3	5 3 2	2	4	5 3 2 4
4	5 3 2 4	4	Ø	5 3 2
5	5 3 2	2	Ø	5 3
6	5 3	3	6	5 3 6
7	5 3 6	6	1	5 3 6 1
8	5 3 6 1	1	Ø	5 3 6
9	5 3 6	6	8	5 3 6 8
10	5 3 6 8	8	9	5 3 6 8 9
11	5 3 6 8 9	9	Ø	5 3 6 8
12	5 3 6 8	8	Ø	5 3 6
13	5 3 6	6	Ø	5 3
14	5 3	3	Ø	5
15	5	5	Ø	Ø

Visited & Trace

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

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

Try some examples, report path from s to v:

PATH(1) → ?

PATH(2) → ?

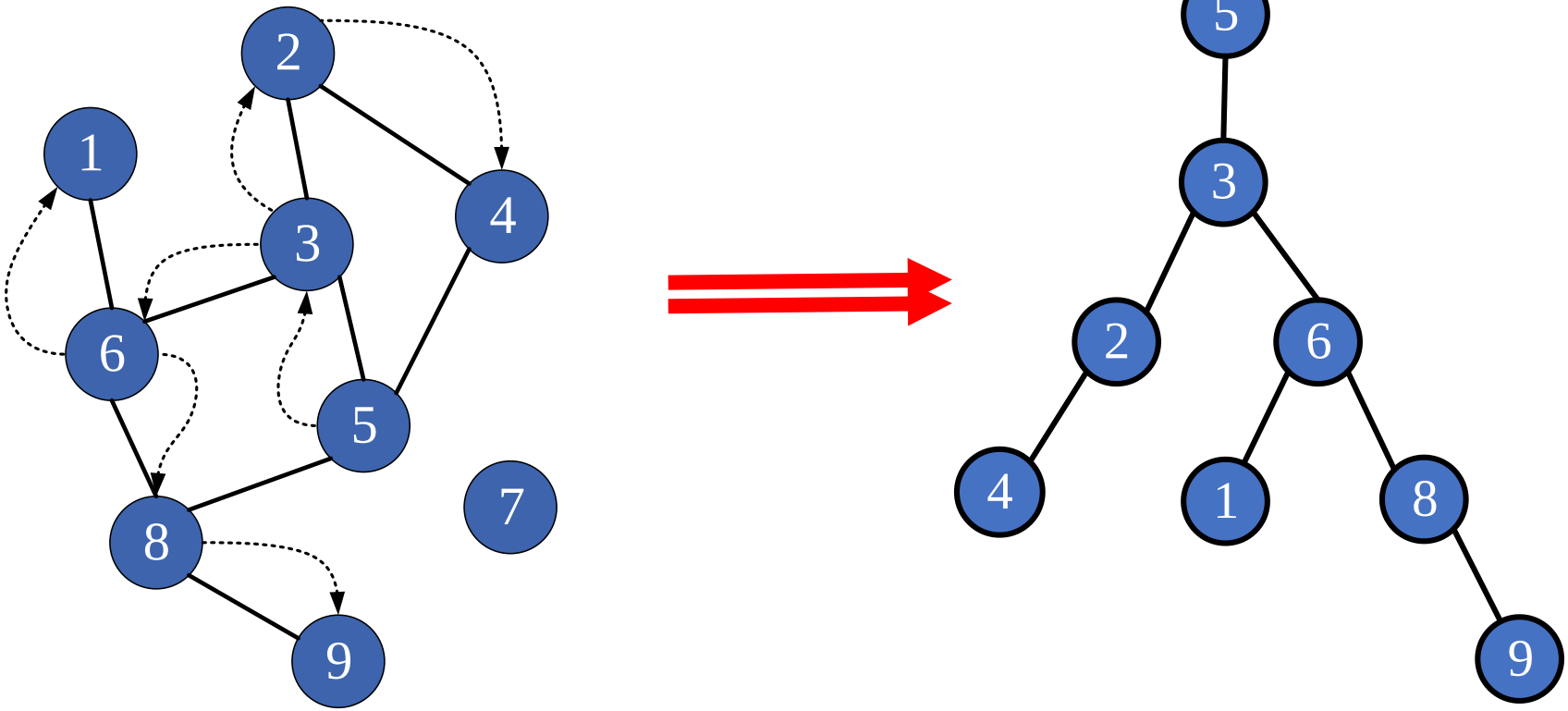
PATH(7)→ ?

DFS Tree

Kết quả DFS-Tree.
Chú ý rằng: DFS-Tree có thể sâu hơn BFS-Tree

Captures the structure of the recursive calls

- when we visit a neighbor k of v, we add k as child of v
- whenever DFS returns from a vertex v, we climb up in the tree from v to its parent



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.

Time Complexity of DFS-Using adjacency list

Running time:

- We never visited a vertex more than once
- We had to examine all edges of the vertices
  - We know $\sum_{vertex\ v} \deg(v) = 2m$ where m is the number of edges
- So, the running time of DFS is proportional to the number of edges and number of vertices (same as BFS)

  O(n + m)

- You will also see this written as:

  O(|v|+|e|)

  |v| = number of vertices (n)

  |e| = number of edges (m).

Applications of Depth First Search

Depth-first search (DFS) is an algorithm (or technique) for traversing a graph.

Following are the problems that use DFS as a building block.

1. For a weighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.

2. Detecting cycle in a graph
   A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.

3. Path Finding
   We can specialize the DFS algorithm to find a path between two given vertices u and z.
   i) Call DFS(G, u) with u as the start vertex.
   ii) Use a stack S to keep track of the path between the start vertex and the current vertex.
   iii) As soon as destination vertex z is encountered, return the path as the contents of the stack

4. Topological Sorting
   Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in makefiles, data serialization, and resolving symbol dependencies in linkers [2].

5. To test if a graph is bipartite
   We can augment either BFS or DFS when we first discover a new vertex, color it opposited its parents, and for each other edge, check it doesn't link two vertices of the same color. The first vertex in any connected component can be red or black!

6. Finding Strongly Connected Components of a graph A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.

7. Solving puzzles with only one solution, such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.)

Nhận xét DFS: 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.

Chúc các em học tốt!