Graph Traversal (Graph Search)

- A traversal is a systematic procedure for exploring a graph by examining all of its vertices and edges.

- **Applications of graph search**:
  - Web Crawling
  - Social Networking
  - Network Broadcast
  - Base for other algorithms
  - … and more

- For example: a Web spider, or crawler, which is the data collecting part of a search engine, must explore a graph of hypertext documents by examining its vertices, which are the documents, and its edges, which are the hyperlinks between documents.

Graph Traversal Algorithms

- **Breadth First Search Algorithm (BFS)**:
  - Start several paths at a time and advance in each step at a time
- **Depth First Search Algorithm (DFS)**:
  - Once a path is found, continue the search until the end of the path.

Breadth First Search

- Given a graph G=(V,E) and a distinguished ***source*** vertex **s**, breadth-first search
  - Systematically explores the edges of G to "discover" every vertex that is reachable from s.
  - It computes the distance (smallest number of edges) from s to each reachable vertex.
  - Finds a shortest path from s to every other vertex in G.
  - It also produces a "breadth-first tree" with root s that contains all reachable vertices. It computes a breadth-first forest if G is not connected (or possibly in directed graph).
  - Determines (Check) whether G is connected or not.
  - Computes the connected components of G.

BFS Algorithm Pseudocode

$\boldsymbol{Input}$: $A\ graph\ G = (V, E)\ and\ a\ start\ vertex\ s.$

$For\ Each\ v \in V$

$\quad dist(v) := \infty, par(v) := null\ and\ v.vsisted := False.$

$Create\ a\ new\ empty\ queue\ Q.$

$dist(s) := 0\ and\ s.visited := True.$

$Q.enqueue(s)$

$While\ Q\ is\ not\ empty$

$\quad v := Q.dequeue()$

$\quad For\ Each\ u \in adj(v)$

$\quad\quad If\ u.visited == False$

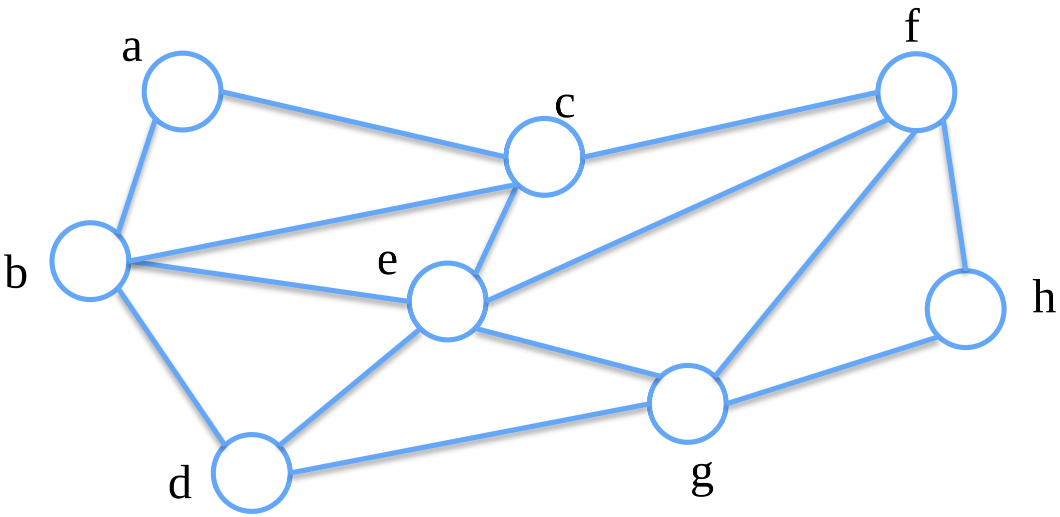
$\quad\quad\quad dist(u) := dist(v) + 1.$

$\quad\quad\quad par(u) := v$

$\quad\quad\quad u.visited := True$

$\quad\quad\quad Q.enqueue(u)$

BFS Algorithm

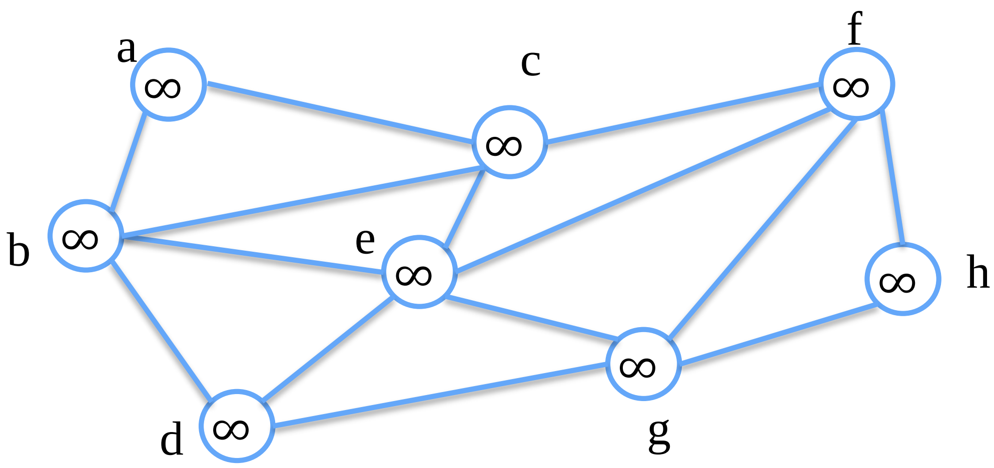


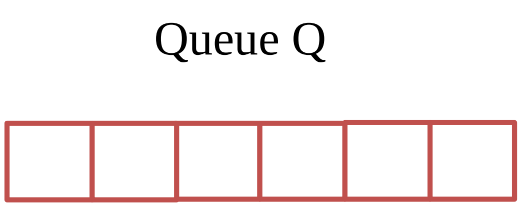
BFS Algorithm

Preparation (Initialization)

For every vertex v,

set the distance dist(v) := ∞ and the parent $parent$(v) := null, v.visited=False.





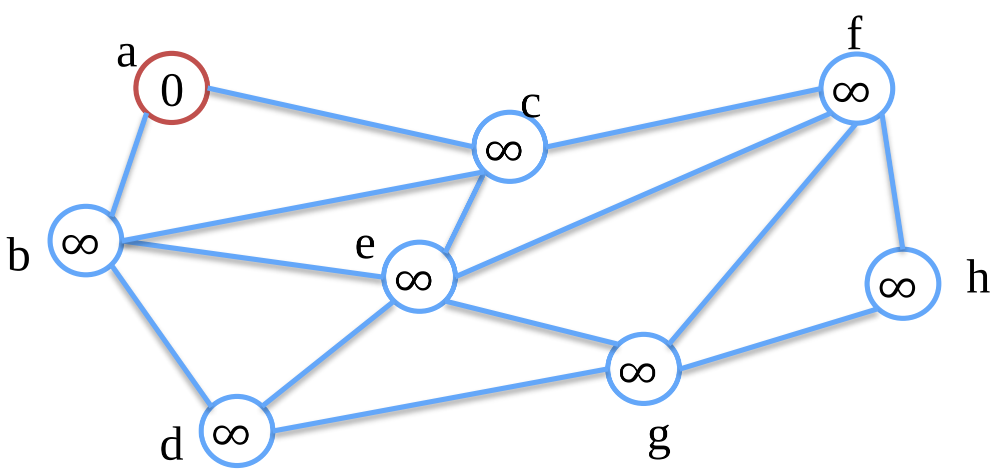
BFS Algorithm

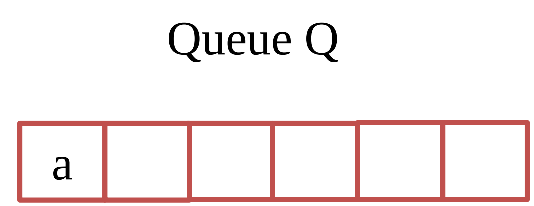
Preparation (Initialization)

Let us say our starting vertex is a, so

dist(a)=0, a.visited=True \\visited vertex colored red

Add s to queue Q.





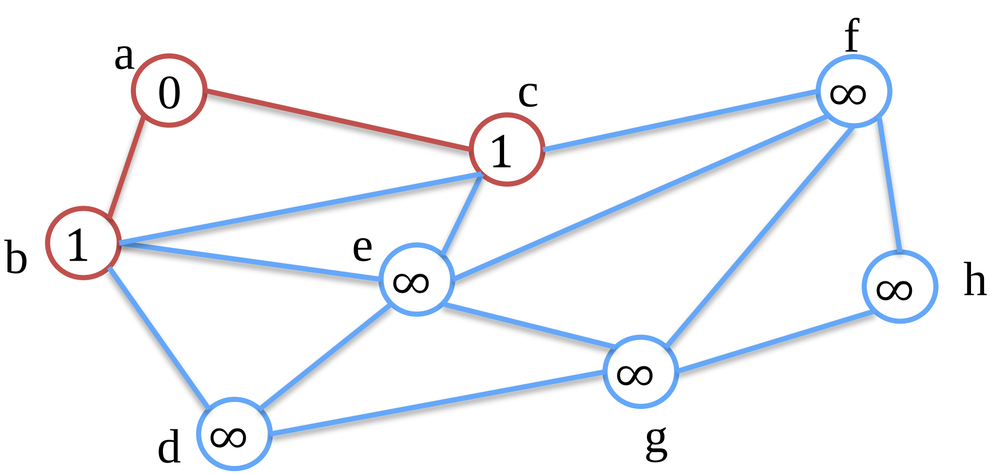
BFS Algorithm

Queue Q

a					

Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove *the vertex from Q*
$For\ each\ u \in Adj(v)$
    if u.visited==False
        $dist(u) := \ dist(v) + \ 1$
        $par(u) := \ v$
        u.visited=True
    Q.enqueue(u) *add u to Q*.

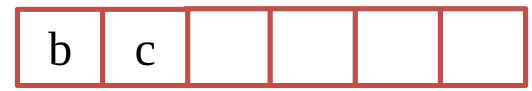


Queue Q

b	c				

BFS Algorithm

Queue Q



Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove *the vertex from Q*

$For\ each\ u \in Adj(v)$

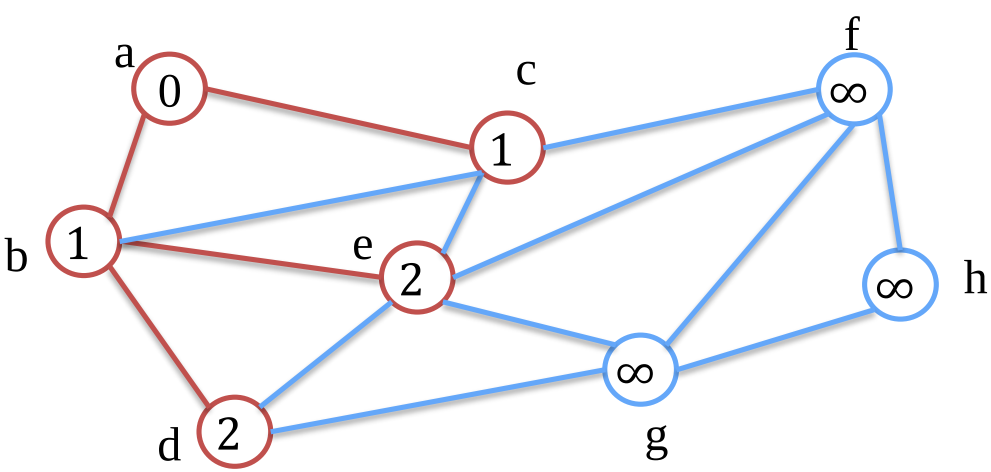
if u.visited==False

$dist(u) := dist(v) + 1$

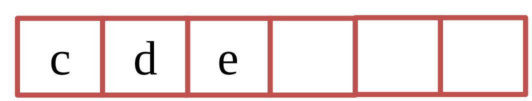
$par(u) := v$

u.visited=True

Q.enqueue(u) *add u to Q*.



Queue Q



BFS Algorithm

Queue Q

c	d	e			

Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove *the vertex from Q*

$For\ each\ u \in Adj(v)$

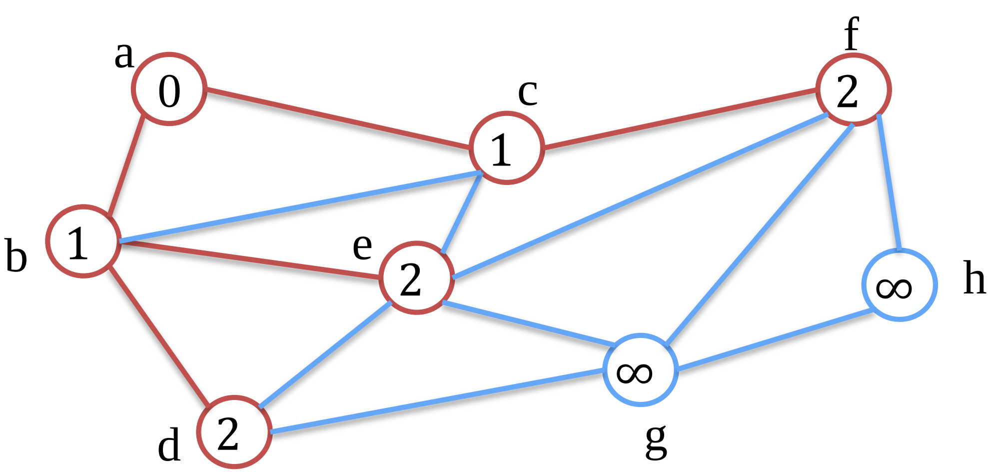
if u.visited==False

$dist(u) := \ dist(v) + \ 1$

$par(u) := \ v$

u.visited=True

Q.enqueue(u) \\ *add u to Q*.



Queue Q

d	e	f			

BFS Algorithm

Queue Q

c	d	e			

Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove *the vertex from Q*

$For\ each\ u \in Adj(v)$

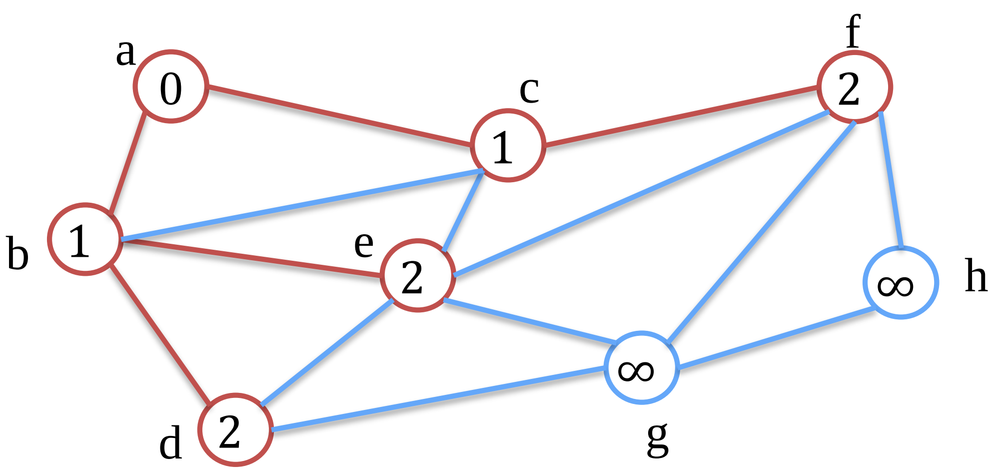
if u.visited==False

$dist(u) := \ dist(v) + \ 1$

$par(u) := \ v$

u.visited=True

Q.enqueue(u) *add u to Q*.



Queue Q

d	e	f			

BFS Algorithm

Queue Q

d	e	f			

Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove *the vertex from Q*

$For\ each\ u \in Adj(v)$

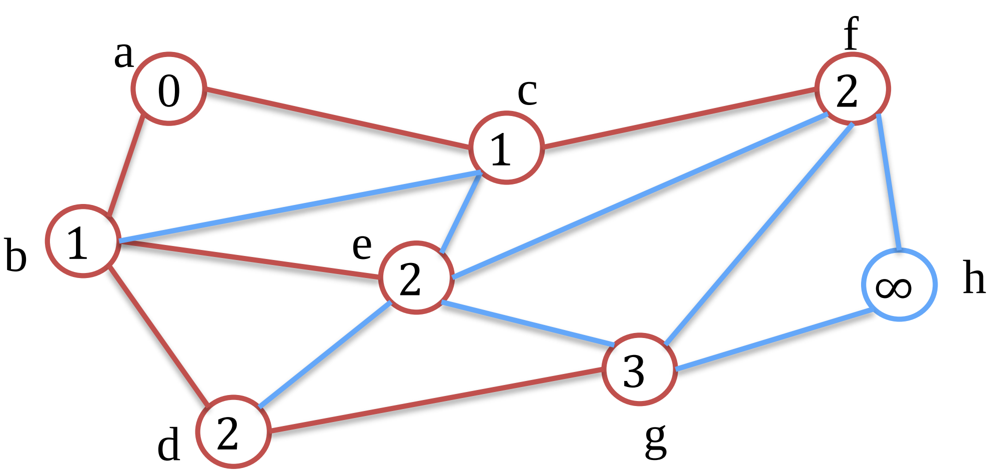
if u.visited==False

$dist(u) := dist(v) + 1$

$par(u) := v$

u.visited=True

Q.enqueue(u) *add u to Q*.



Queue Q

e	f	g			

BFS Algorithm

Queue Q

e	f	g			

Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove $the\ vertex\ from\ Q$

$For\ each\ u \in Adj(v)$

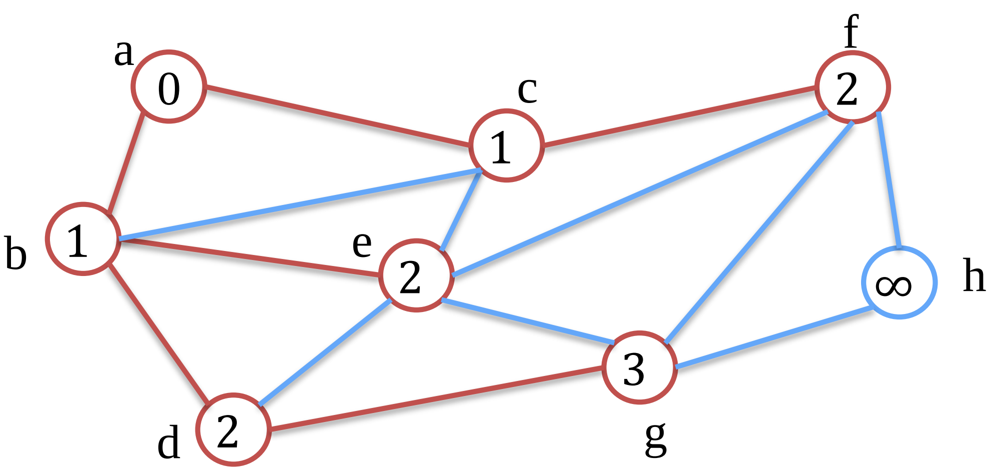
if u.visited==False

$dist(u) := dist(v) + 1$

$par(u) := v$

u.visited=True

Q.enqueue(u) \\$add\ u\ to\ Q$.

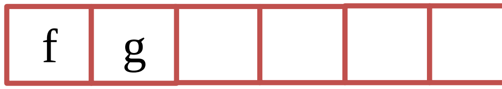


Queue Q

f	g				

BFS Algorithm

Queue Q



Iterations (While Q is not Empty)

$v := Q.dequeue(\ \ )$ \\Remove *the vertex from Q*

For each $u \in Adj(v)$

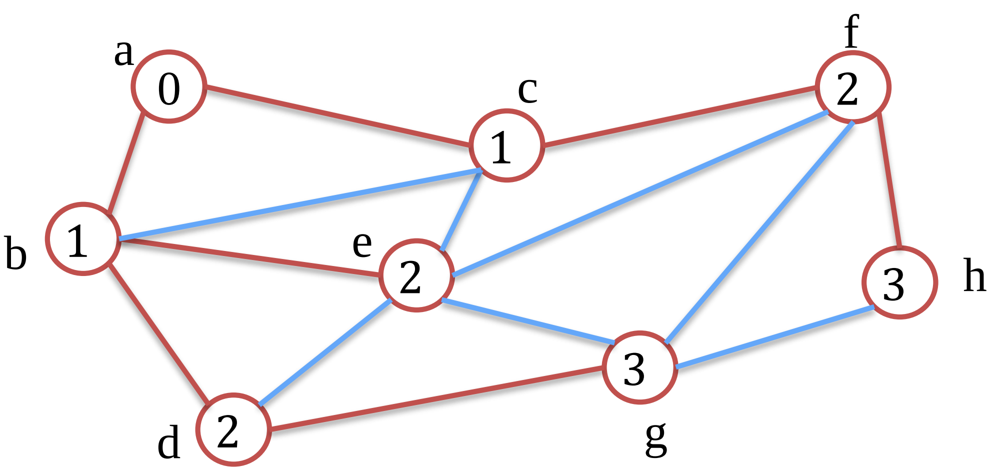
if u.visited==False

$dist(u) := dist(v) + 1$

$par(u) := v$

u.visited=True

Q.enqueue(u) *add u to Q*.



Queue Q

BFS Algorithm

Queue Q

g	h				

Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove $the\ vertex\ from\ Q$

$For\ each\ u \in Adj(v)$

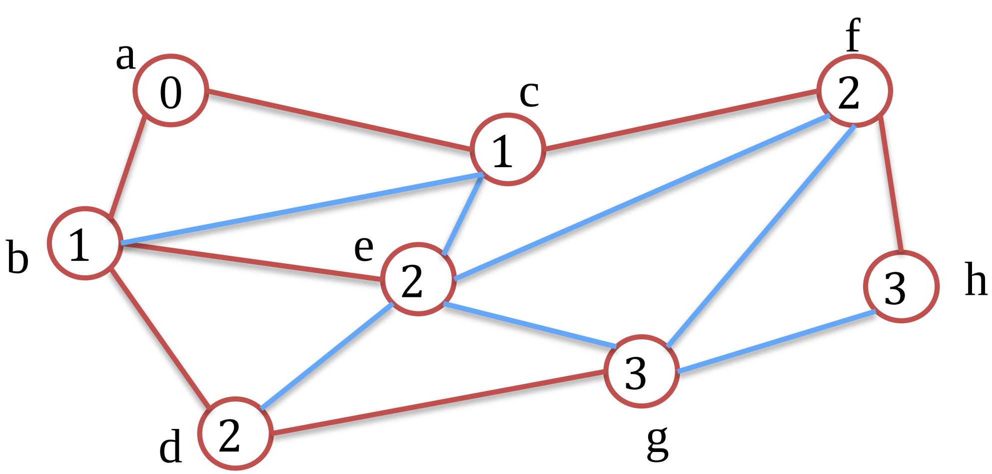
if u.visited==False

$dist(u) := dist(v) + 1$

$par(u) := v$

u.visited=True

Q.enqueue(u) \\$add\ u\ to\ Q.$

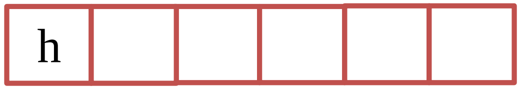


Queue Q

h					

BFS Algorithm

Queue Q



Iterations (While Q is not Empty)

$v := Q.dequeue(\quad)$ \\Remove *the vertex from Q*

For each $u \in Adj(v)$

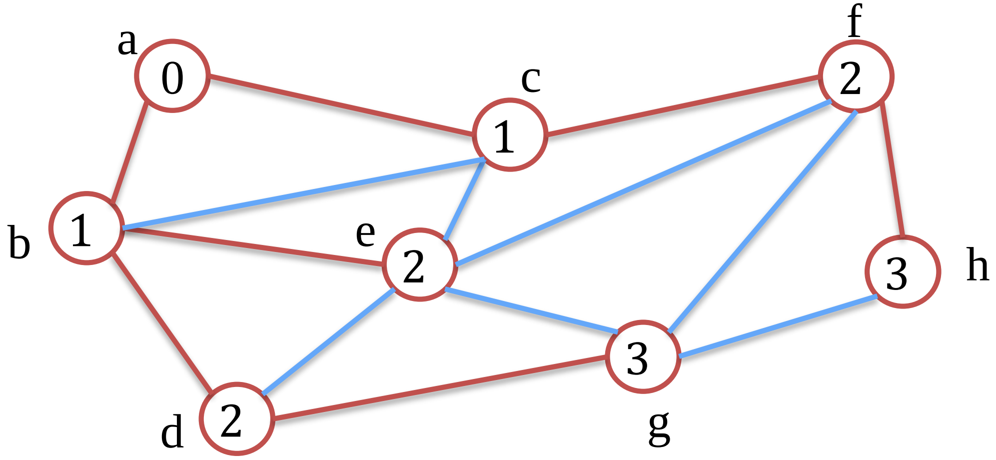
  if u.visited==False

    $dist(u) := dist(v) + 1$

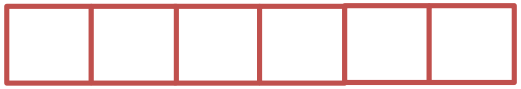
    $par(u) := v$

    u.visited=True

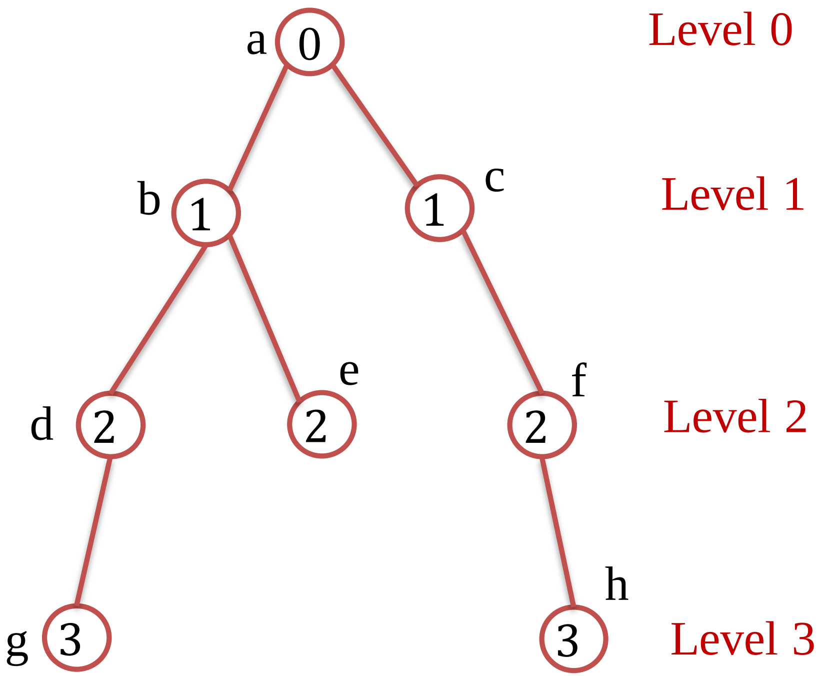
    Q.enqueue(u) *add u to Q.*

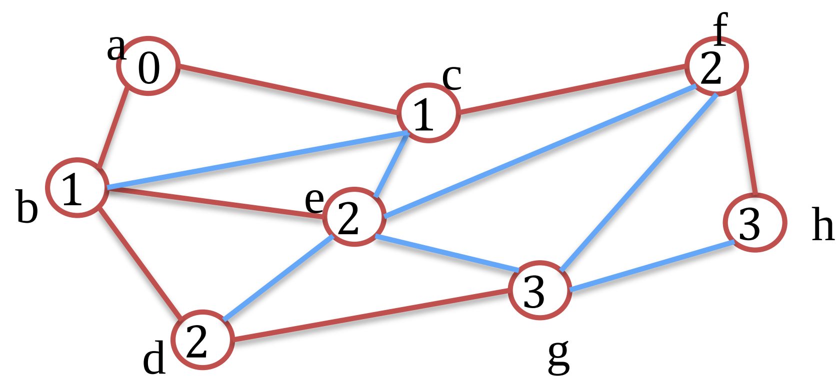


Queue Q



BFS Algorithm





- BFS Tree
- Computes the distance from s to each reachable vertex.
- Finds a shortest path from s to every other vertex in G.

BFS Algorithm Running time

- Let G be a graph with **n** vertices and **m** edges represented with the **adjacency list**.
- BFS on a graph with ***n*** vertices and ***m*** edges takes $O(n + m)$ time.
- Analysis:
  - The initialization process takes $O(n)$ time.
  - The operations of enqueuing and dequeuing take $O(1)$ time, and so the total time devoted to queue operations is $O(n)$ time.
  - The total time spent in scanning adjacency lists is $O(m)$ time, since the sum of the lengths of all the adjacency lists is $\theta(m)$. Recall that $\sum_v \deg(v) = 2m$.

Total runtime: $O(n + m)$

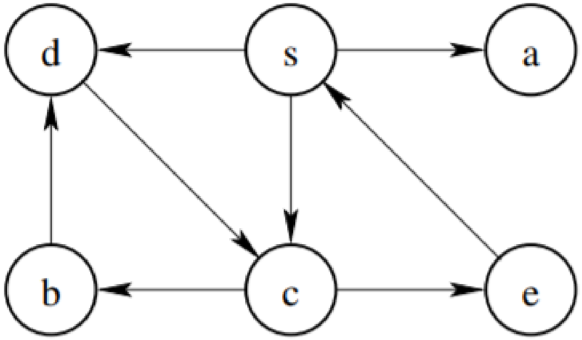
BFS Basics

- The Breadth First Search traversal of a graph will result into **Tree/forest**.
- What is the difference between applying BFS on a graph and a tree?
  Traversal of a graph is different from tree because there can be a loop in graph so we must maintain a visited flag for every vertex.
- The Data structure used in standard implementation of Breadth First Search is Queue.

Exercises

- What is the running time of BFS if we represent its input graph by an adjacency matrix and modify the algorithm to handle this form of input? Analyze the running time.

- Give the visited vertex order on running BFS on the following graph, starting with the vertex s.



Exercises

- Describe the details of an $O(n + m)$ time algorithm for computing all the connected components of an undirected graph G with n vertices and m edges.

Depth First Search DFS Algorithm

- Follow path until you get stuck.
- If got stuck, backtrack path until reach unexplored neighbor.
- Continue on unexplored neighbor.

Depth First Search

- Given a graph G=(V,E) and a distinguished ***source*** vertex **s**, depth-first search
  - Systematically explores the edges of G to "discover" every vertex that is reachable from s.
  - It also produces a "depth-first tree" with root s that contains all reachable vertices. It computed a depth-first forest if G is not connected (or possibly in directed graph).
  - Determines (Check) whether G is connected or not.
  - Computes the connected components of G.
  - Cycle detection

DFS Algorithm Pseudocode

$\boldsymbol{Input}$: $A\ graph\ G = (V, E)$

$For\ Each\ v \in V$

$\quad par(v) := null$

$\quad v.vsisted := False.$

$For\ Each\ v \in V$

$\quad If\ v.visited == False$

$\quad\quad DFS\text{-}visit(v)$

$DFS - visit(v)$

$\quad v.visited := True$

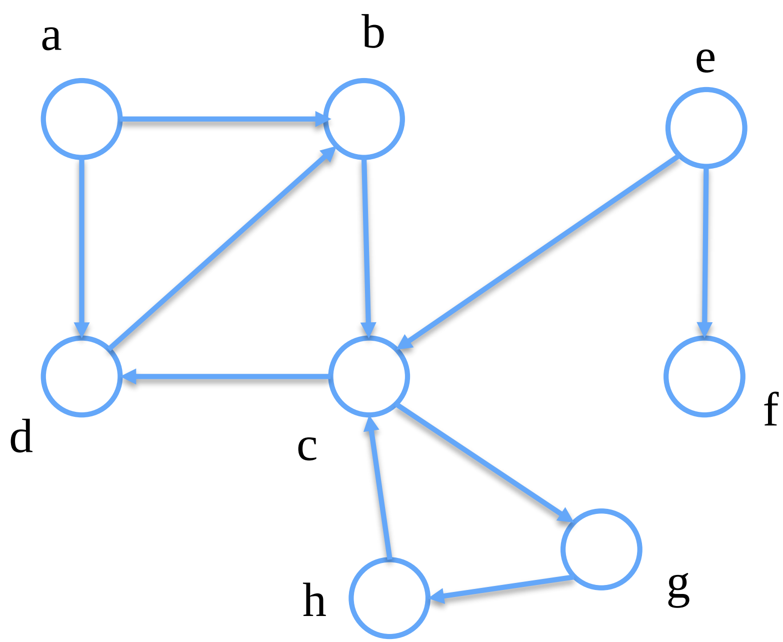
$\quad For\ Each\ u \in adj(v)$

$\quad\quad If\ u.visited == False$

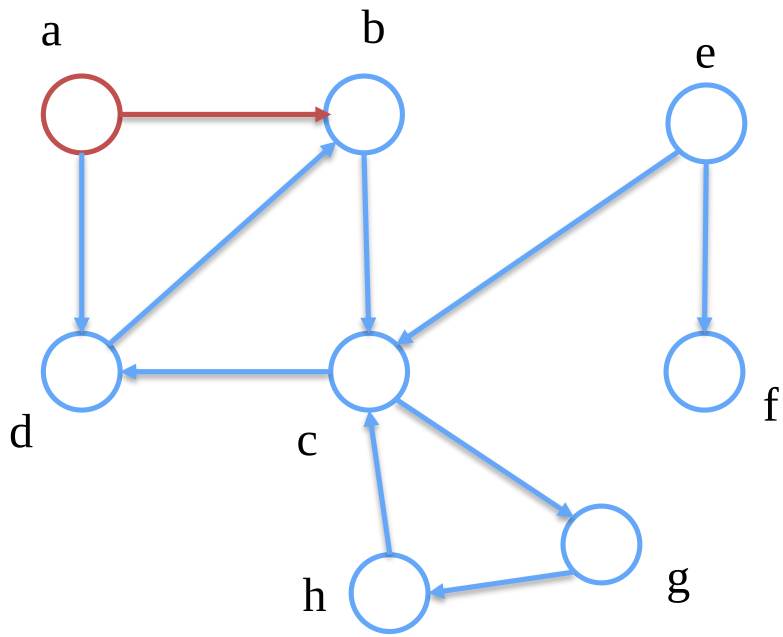
$\quad\quad\quad par(u) := v$

$\quad\quad\quad DFS\text{–}visit(v)$

DFS algorithm

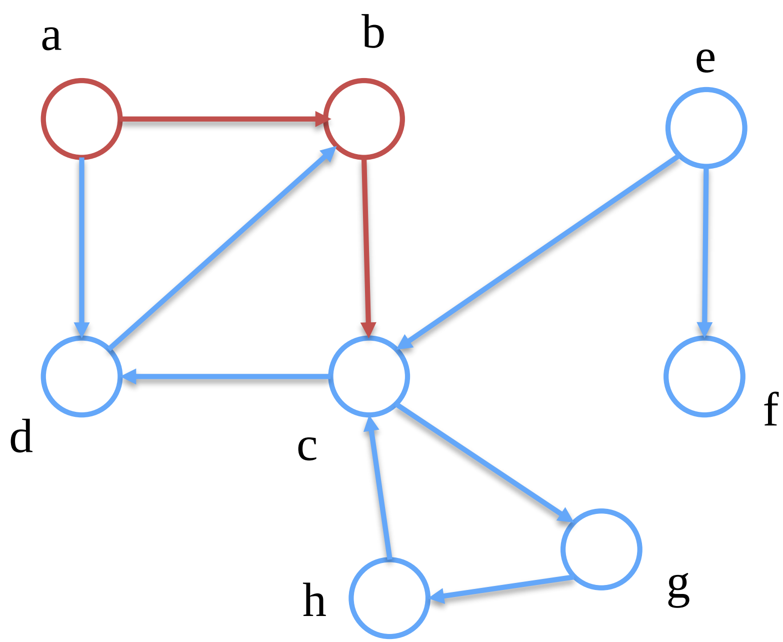


DFS algorithm



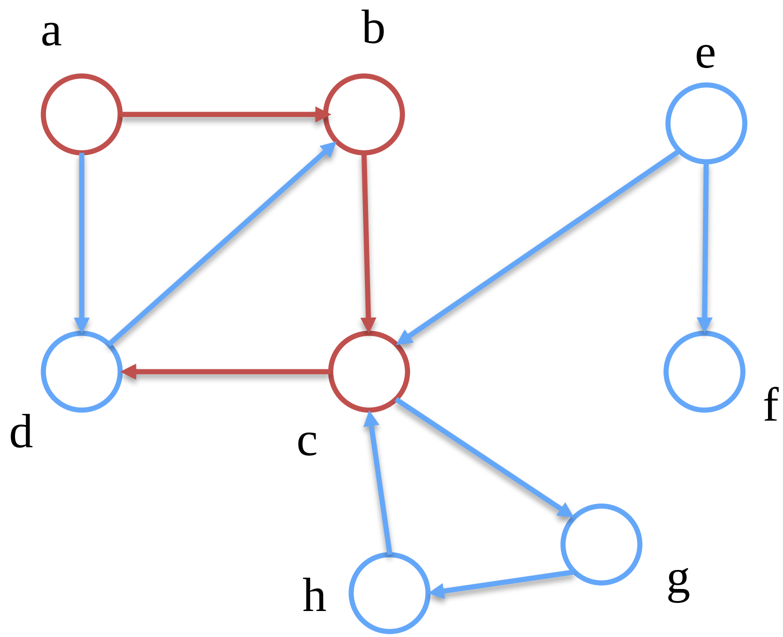
Stack: a

DFS algorithm



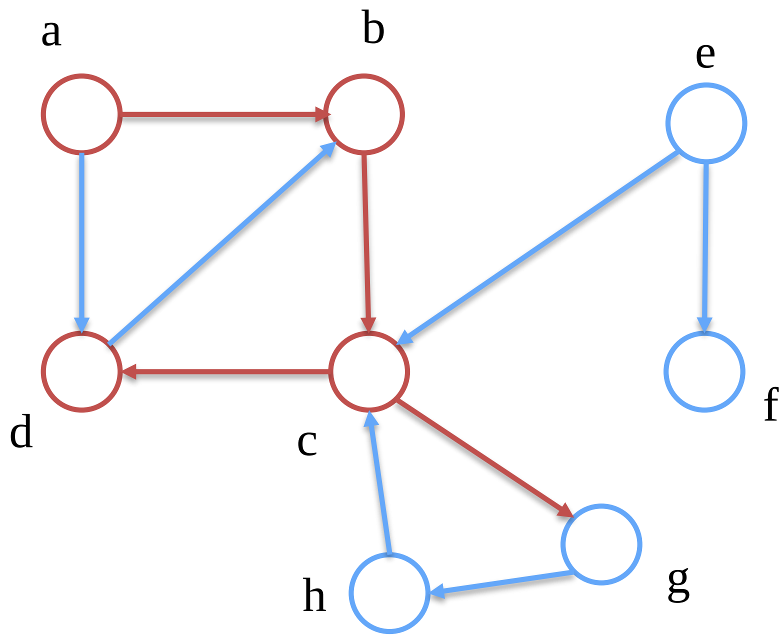
Stack: a b

DFS algorithm



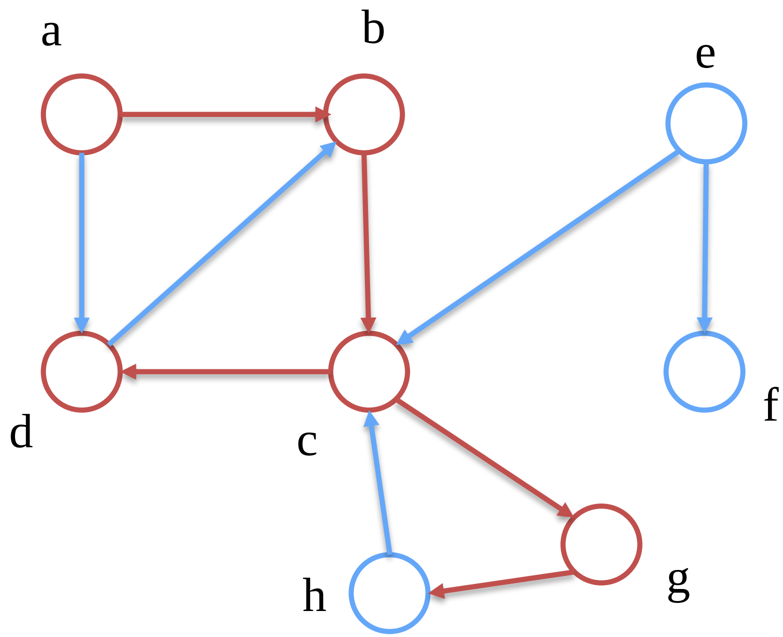
Stack: a b c

DFS algorithm



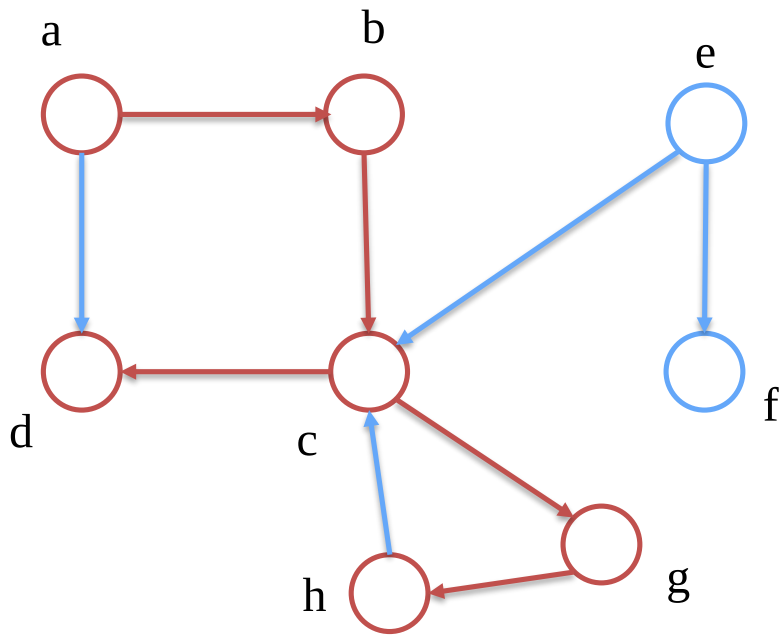
Stack: a b c

DFS algorithm



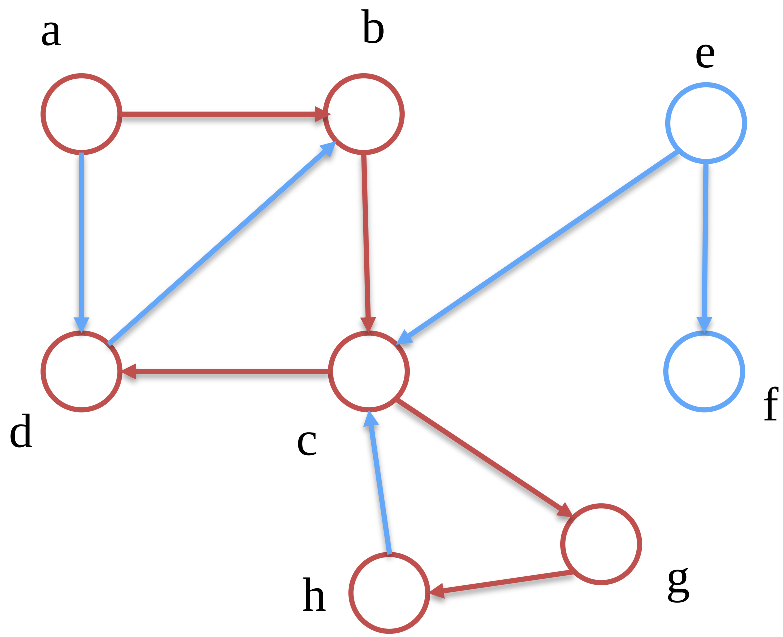
Stack: a b c g

DFS algorithm



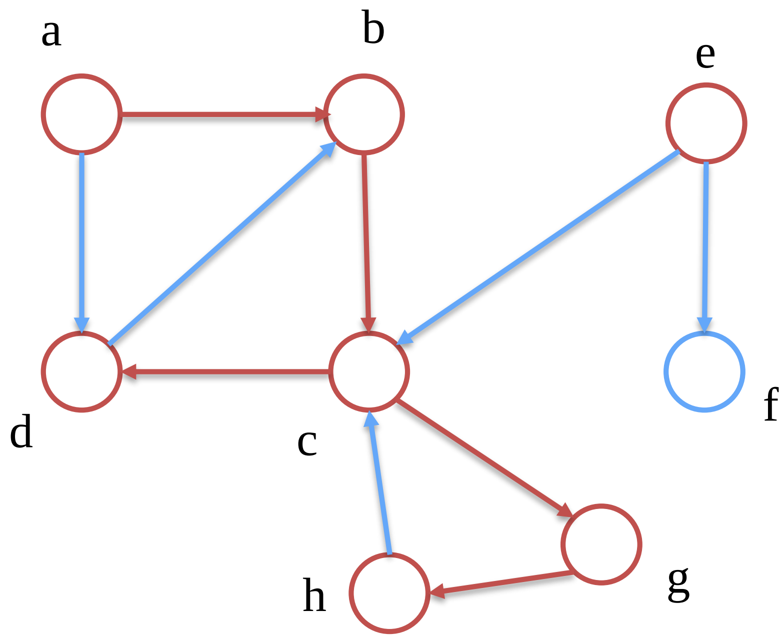
Stack: a b c g

DFS algorithm



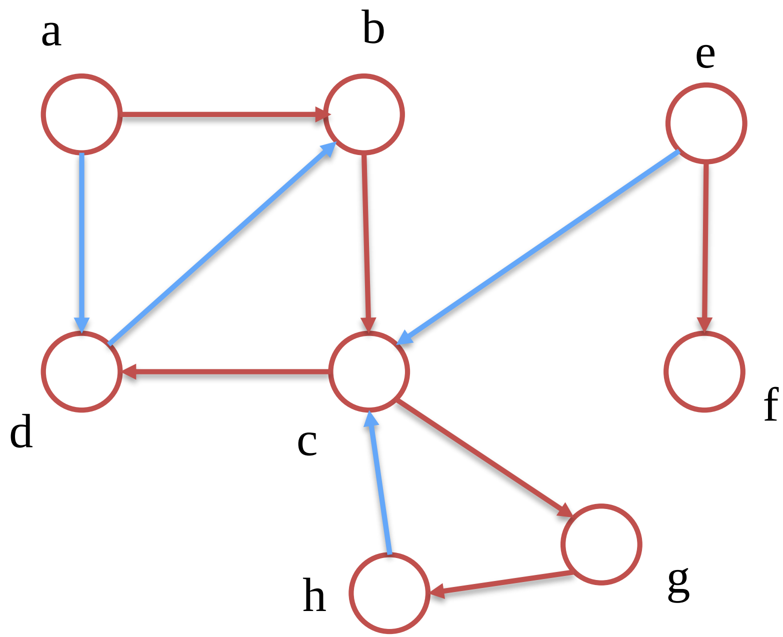
Stack:

DFS algorithm



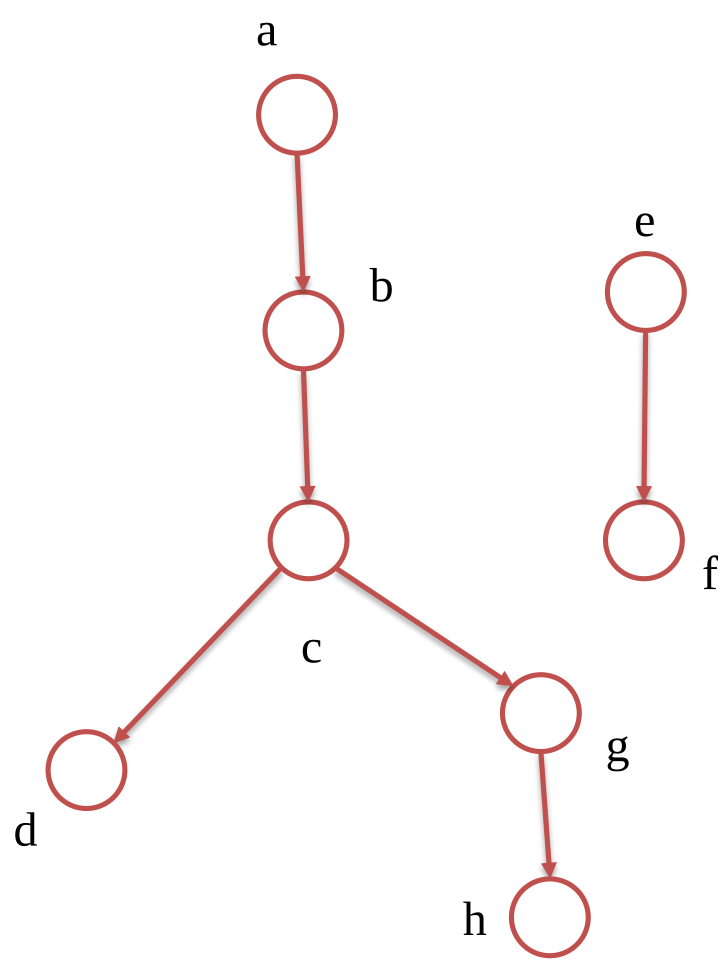
Stack: e

DFS algorithm

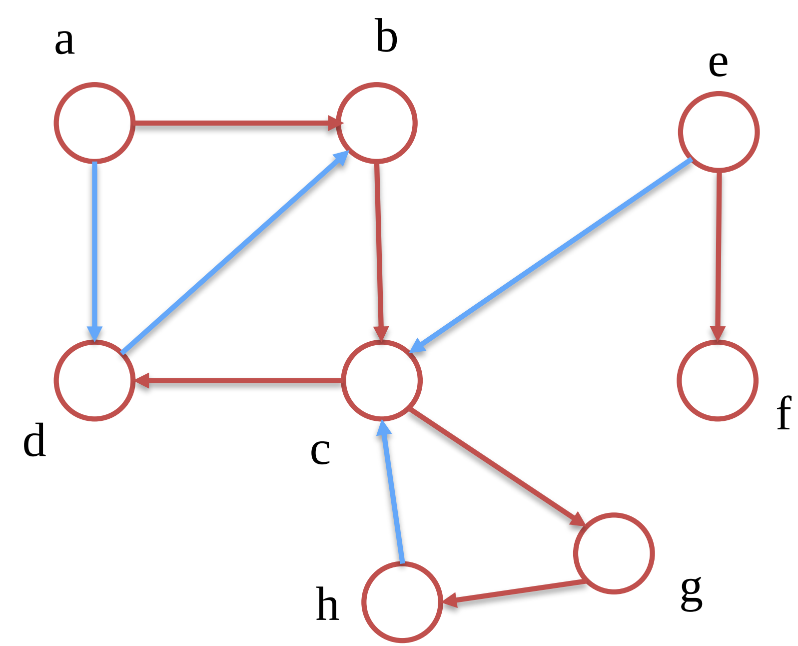


Stack:

DFS algorithm



Depth-First Forest



DFS Running time

- Let G be a graph with **n** vertices and **m** edges represented with the **adjacency list**.
- DFS on a graph with ***n*** vertices and ***m*** edges takes $O(n + m)$ time.
- Analysis:
  - Initialization loop in DFS : O(n)
  - Main loop in DFS: O(n) exclusive of time to execute calls to DFS-VISIT
  - DFS-VISIT is called exactly once for each v∈V. For loop of DFS-VISIT(u) is executed |Adj[u]| time. Since Σ |Adj[u]| = 2E,

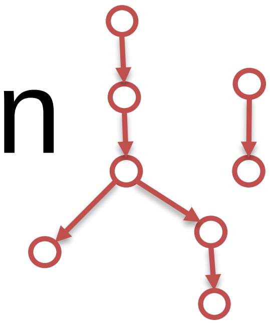
Total runtime: $O(n + m)$

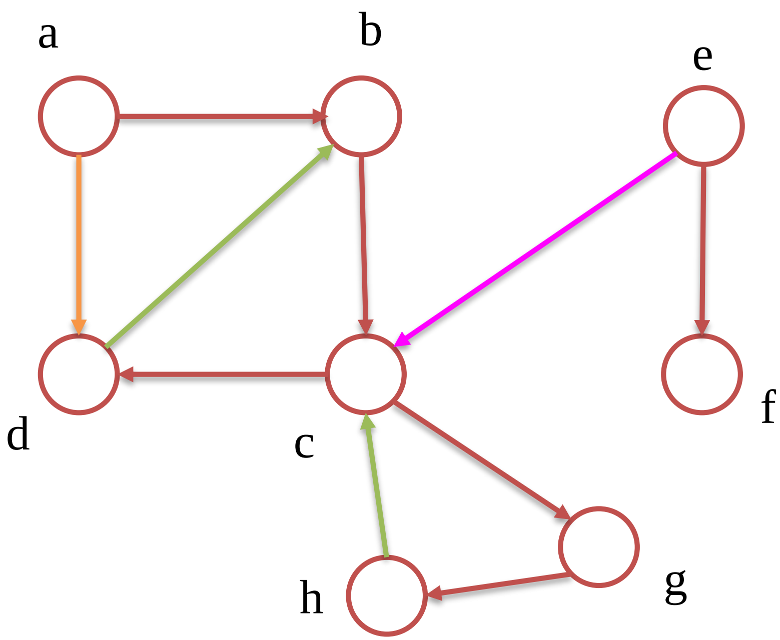
DFS – Edges classification

- A DFS partitions the edges in four groups:

1) Tree edges: Are edges in the depth-first forest.
2) Back edges: Are those nontree edges (u,v) connecting a vertex u to an ancestor v in a depth-first tree.
3) Forward edges (only in directed graphs): are those nontree edges (u,v) connecting a vertex u to a descendant in a depth-first tree.
4) Cross edges (only in directed graphs): Remaining edges.

DFS – Edges classification





Tree Edges

Back Edges

Forward Edges

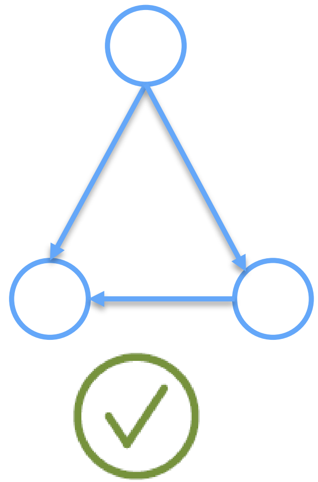
Cross Edges

DFS – Detection Cycles

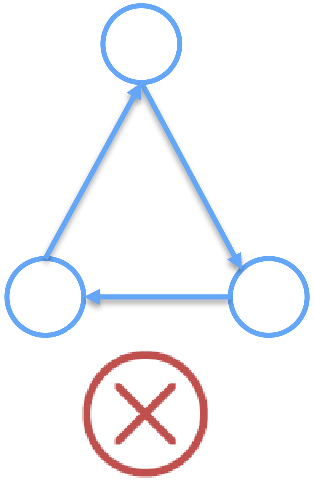
A graph G has a cycle if and only if any DFS has a back edge.

Directed Acyclic Graphs (DAG)

- A *directed acyclic graph* (or DAG for short) is a directed graph that contains no cycles.



It is a DAG,
since there is no cycle

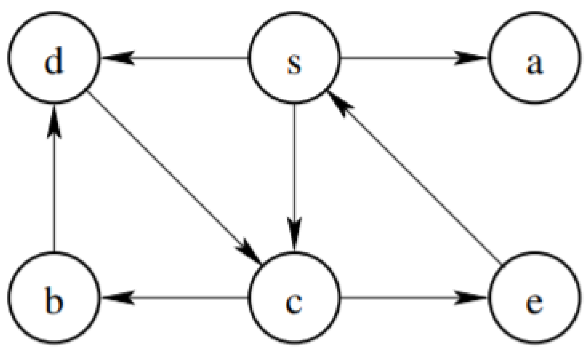


It has a cycle

Exercises

- What is the running time of DFS if we represent its input graph by an adjacency matrix and modify the algorithm to handle this form of input? Analyze the running time.

- Give the visited vertex order on running DFS on the following graph, starting with the vertex s.



Exercises

- Give an $O(|V| + |E|)$ time algorithm to remove all the cycles in a directed graph $G = (V, E)$. Removing a cycle means removing an edge of the cycle. If there are k cycles in G, the algorithm should only remove at most $O(k)$ edges.

- How fast we can check if a directed graph is a DAG or not? Explain.

DFS vs. BFS

Applications	DFS	BFS
Tree / forest, connected components	√	√
Shortest paths and distances		√
cycles	√	