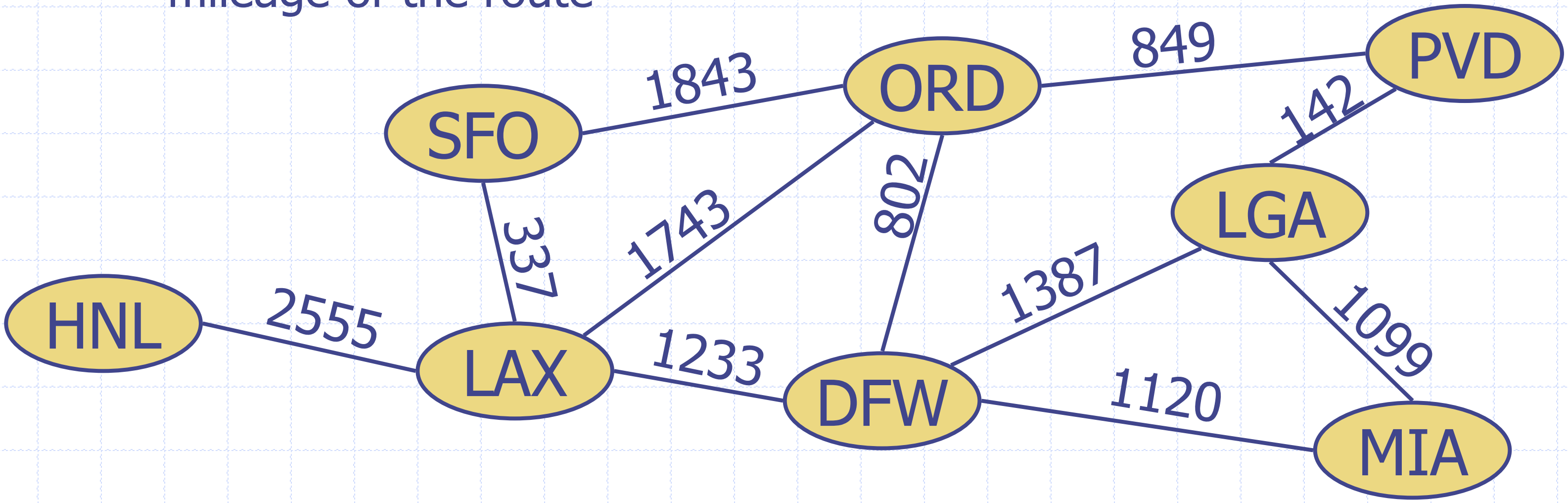
Lecture 11
Graphs, DFS, BFS, topological sort

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

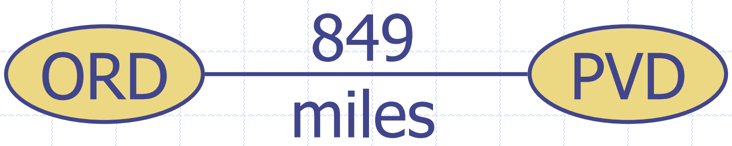
Graphs

- A graph is a pair (V, E), where
  - V is a set of nodes, called vertices
  - E is a collection of pairs of vertices, called edges
  - Vertices and edges are positions and store elements
- Example:
  - A vertex represents an airport and stores the three-letter airport code
  - An edge represents a flight route between two airports and stores the mileage of the route



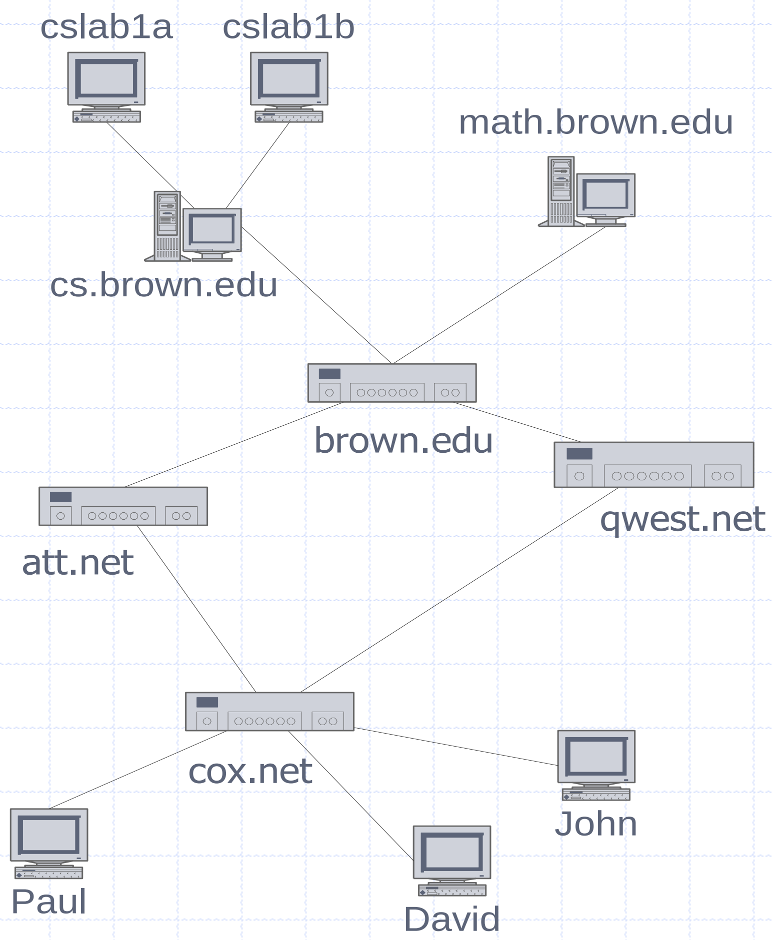
Edge Types

- Directed edge
  - ordered pair of vertices (u,v)
  - first vertex u is the origin
  - second vertex v is the destination
  - e.g., a flight
- Undirected edge
  - unordered pair of vertices (u,v)
  - e.g., a flight route
- Directed graph
  - all the edges are directed
  - e.g., route network
- Undirected graph
  - all the edges are undirected
  - e.g., flight network



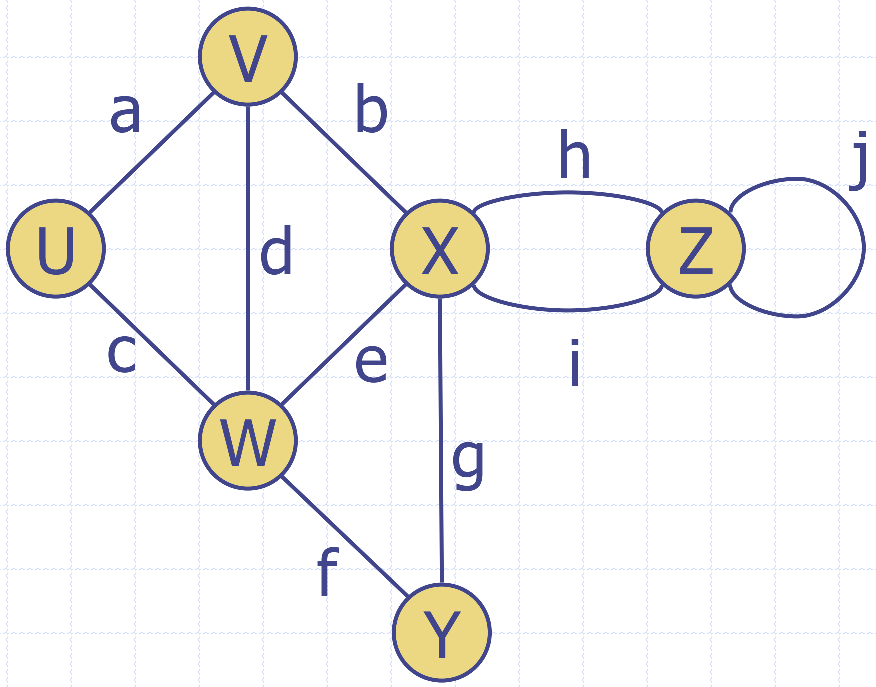
Applications

- Electronic circuits
  - Printed circuit board
  - Integrated circuit
- Transportation networks
  - Highway network
  - Flight network
- Computer networks
  - Local area network
  - Internet
  - Web
- Databases
  - Entity-relationship diagram



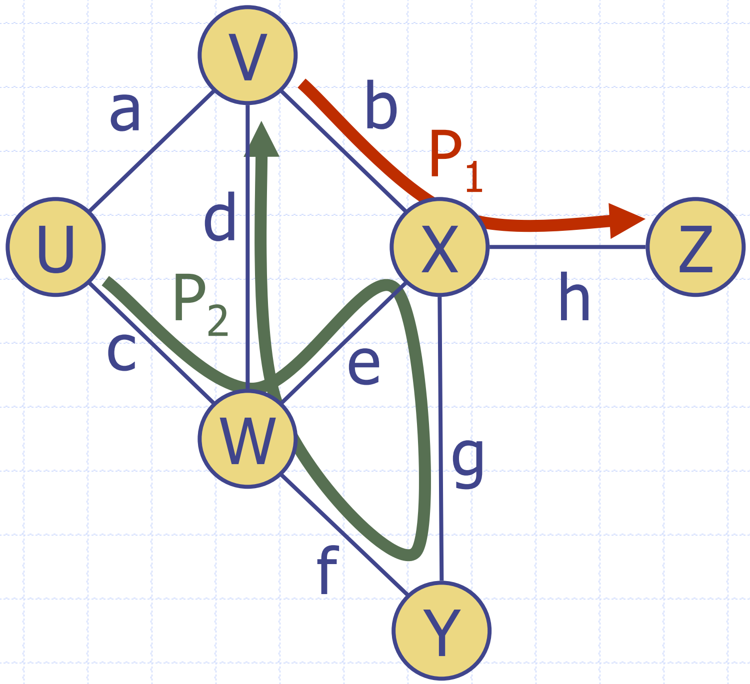
Terminology

- End vertices (or endpoints) of an edge
  - U and V are the endpoints of a
- Edges incident on a vertex
  - a, d, and b are incident on V
- Adjacent vertices
  - U and V are adjacent
- Degree of a vertex
  - X has degree 5
- Parallel edges
  - h and i are parallel edges
- Self-loop
  - j is a self-loop



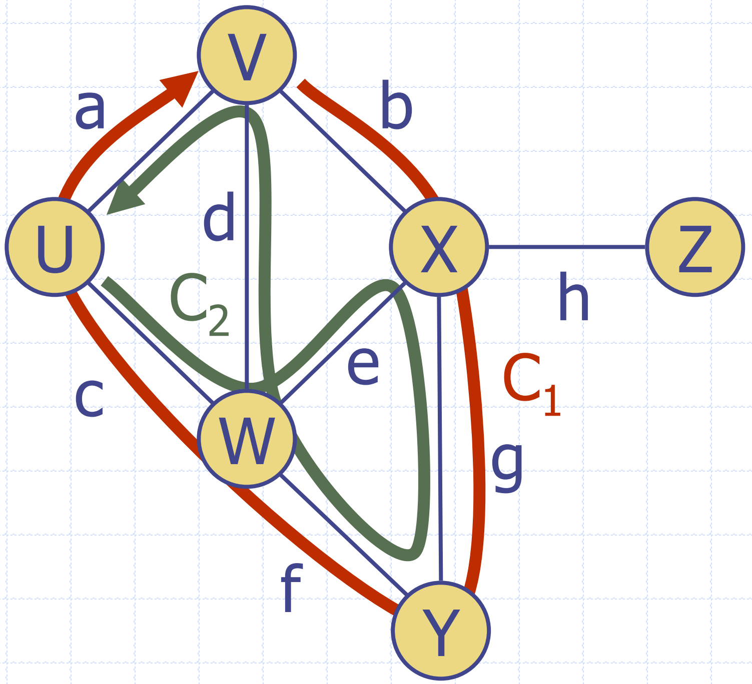
Terminology (cont.)

- Path
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints
- Simple path
  - path such that all its vertices and edges are distinct
- Examples
  - P_1=(V,b,X,h,Z) is a simple path
  - P_2=(U,c,W,e,X,g,Y,f,W,d,V) is a path that is not simple



Terminology (cont.)

- Cycle
  - circular sequence of alternating vertices and edges
  - each edge is preceded and followed by its endpoints
- Simple cycle
  - cycle such that all its vertices and edges are distinct
- Examples
  - C_1=(V,b,X,g,Y,f,W,c,U,a,↵) is a simple cycle
  - C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,↵) is a cycle that is not simple



Properties

Property 1

$\Sigma_v \deg(v) = 2m$

Proof: each edge is counted twice

Property 2

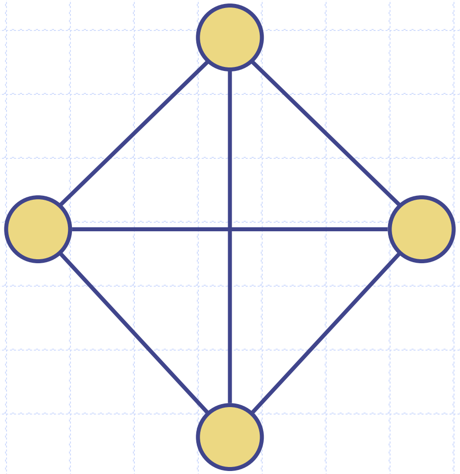
In an undirected graph with no self-loops and no multiple edges

$m \leq n\,(n-1)/2$

Proof: each vertex has degree at most $(n-1)$

Notation

n	number of vertices
m	number of edges
$\deg(v)$	degree of vertex v



Example

- $n = 4$
- $m = 6$
- $\deg(v) = 3$

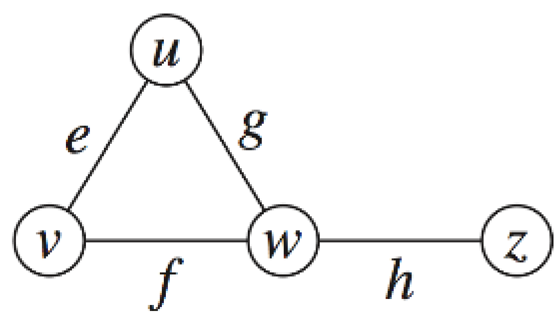
What is the bound for a directed graph?

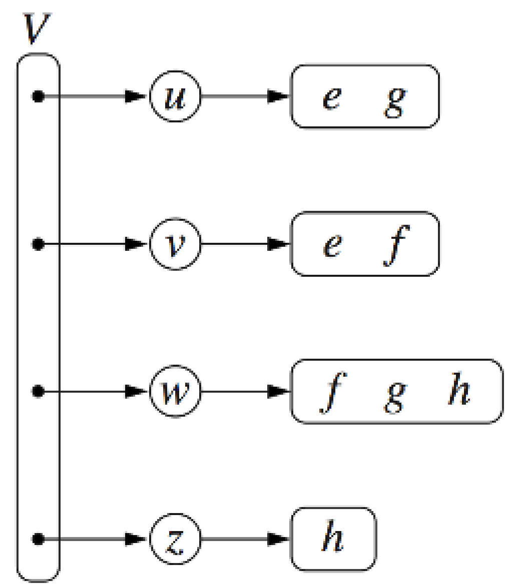
Vertices and Edges

- A **graph** is a collection of **vertices** and **edges**.
- A **Vertex** is can be an abstract unlabeled object or it can be labeled (e.g., with an integer number or an airport code) or it can store other objects
- An **Edge** can likewise be an abstract unlabeled object or it can be labeled (e.g., a flight number, travel distance, cost), or it can also store other objects.

Adjacency List Structure

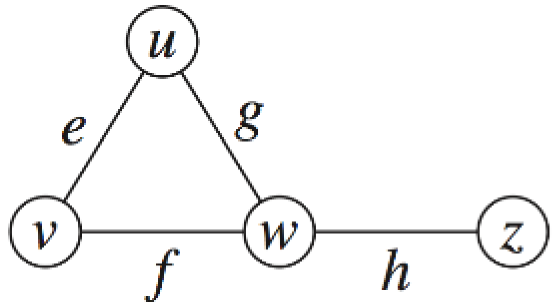
- Incidence sequence for each vertex
  - sequence of references to edge objects of incident edges
- Augmented edge objects
  - references to associated positions in incidence sequences of end vertices

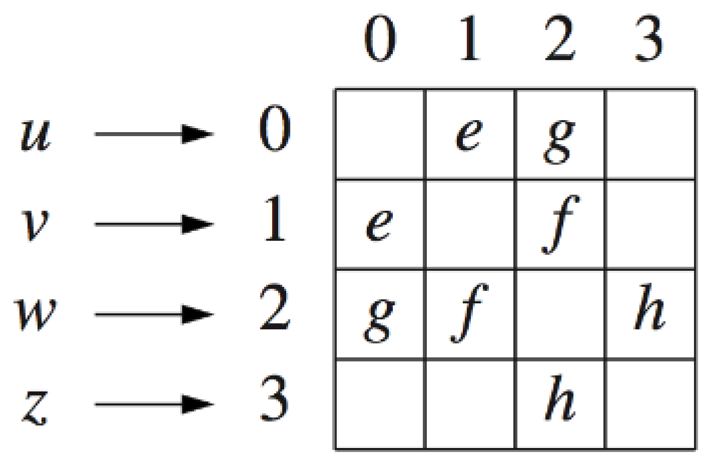




Adjacency Matrix Structure

- Edge list structure
- Augmented vertex objects
  - Integer key (index) associated with vertex
- 2D-array adjacency array
  - Reference to edge object for adjacent vertices
  - Null for non nonadjacent vertices
- The "old fashioned" version just has 0 for no edge and 1 for edge





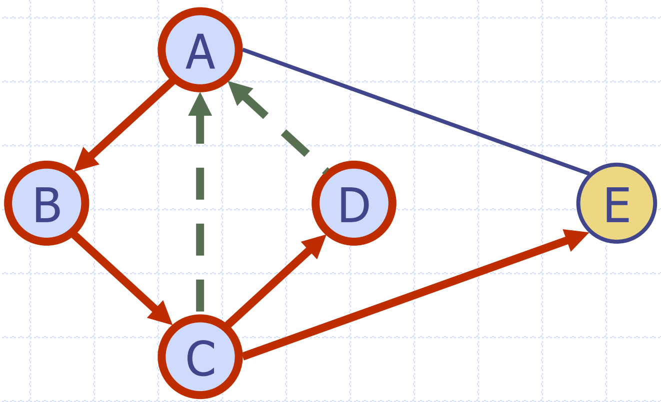
Performance

(All bounds are big-oh running times, except for "Space")

▪ n vertices, m edges ▪ no parallel edges ▪ no self-loops	Adjacency List	Adjacency Matrix
Space	$n + m$	n^2
incidentEdges(v)	$\deg(v)$	n
areAdjacent (v, w)	$\min(\deg(v), \deg(w))$	1
insertVertex(o)	1	n^2
insertEdge(v, w, o)	1	1
removeVertex(v)	$\deg(v)$	n^2
removeEdge(e)	1	1

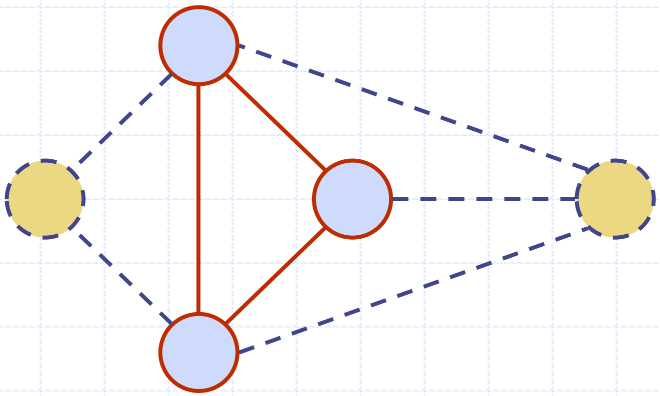
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Depth-First Search

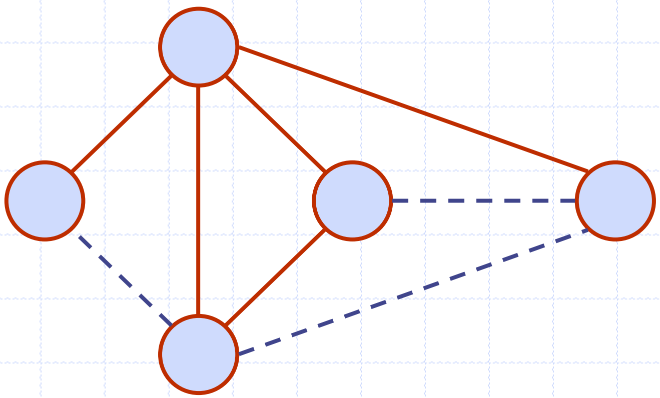


Subgraphs

- A subgraph S of a graph G is a graph such that
  - The vertices of S are a subset of the vertices of G
  - The edges of S are a subset of the edges of G
- A spanning subgraph of G is a subgraph that contains all the vertices of G



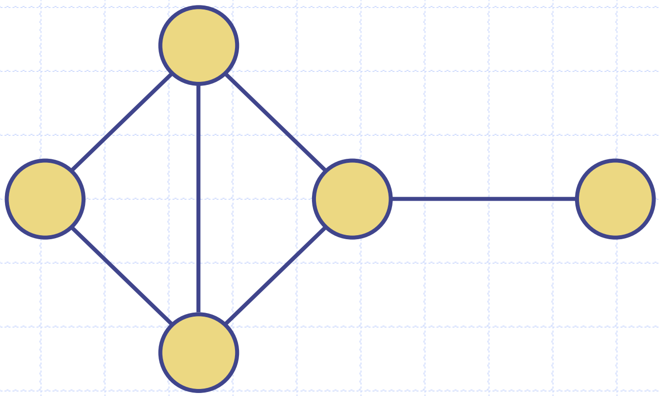
Subgraph



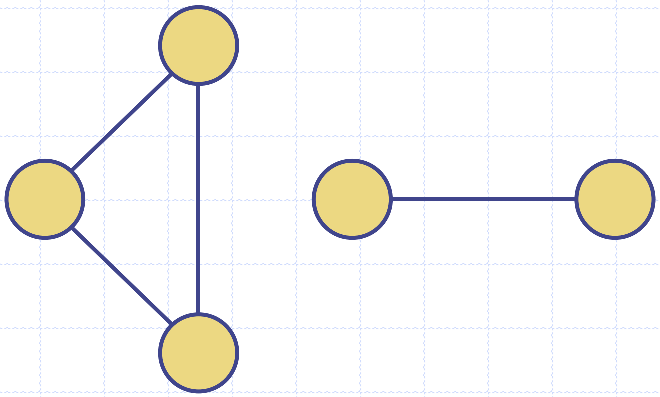
Spanning subgraph

Connectivity

- A graph is connected if there is a path between every pair of vertices
- A connected component of a graph G is a maximal connected subgraph of G



Connected graph

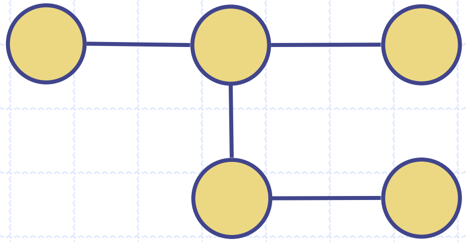


Non connected graph with two connected components

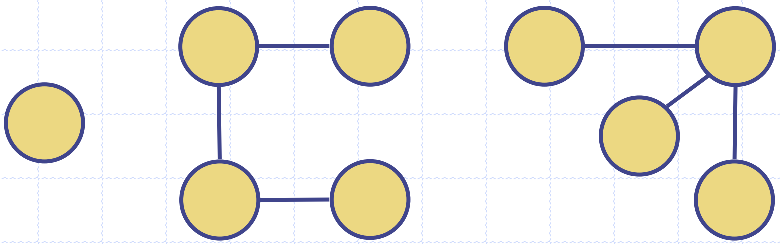
Trees and Forests

- A (free) tree is an undirected graph T such that
  - T is connected
  - T has no cycles

  This definition of tree is different from the one of a rooted tree
- A forest is an undirected graph without cycles
- The connected components of a forest are trees



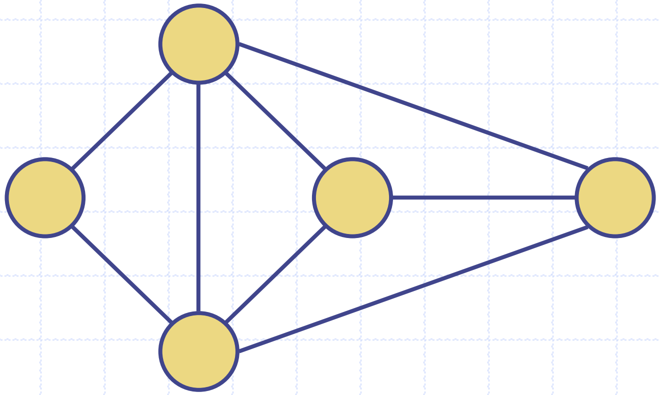
Tree



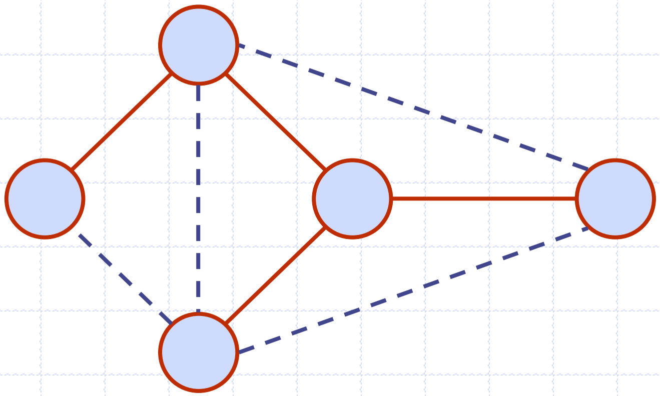
Forest

Spanning Trees and Forests

- A spanning tree of a connected graph is a spanning subgraph that is a tree
- A spanning tree is not unique unless the graph is a tree
- Spanning trees have applications to the design of communication networks
- A spanning forest of a graph is a spanning subgraph that is a forest



Graph



Spanning tree

Depth-First Search

- Depth-first search (DFS) is a general technique for traversing a graph
- A DFS traversal of a graph G
  - Visits all the vertices and edges of G
  - Determines whether G is connected
  - Computes the connected components of G
  - Computes a spanning forest of G
- DFS on a graph with n vertices and m edges takes $O(n + m)$ time
- DFS can be further extended to solve other graph problems
  - Find and report a path between two given vertices
  - Find a cycle in the graph
- Depth-first search is to graphs what Euler tour is to binary trees

DFS Algorithm from a Vertex

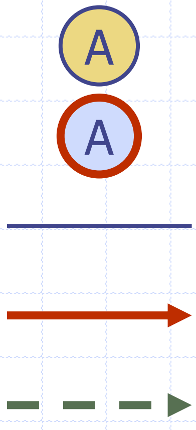
Algorithm DFS(G, v):

Input: A graph G and a vertex v in G

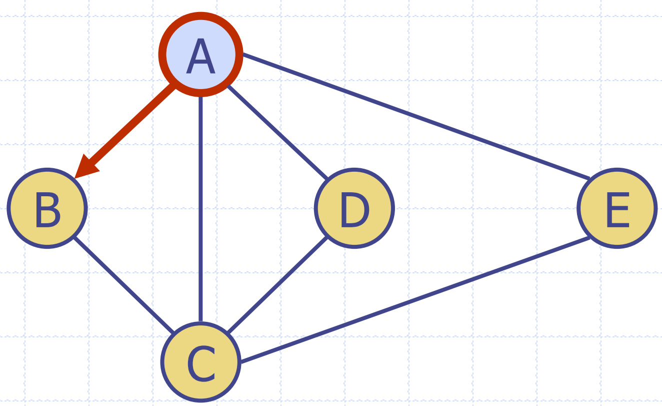
Output: A labeling of the edges in the connected component of v as discovery edges and back edges, and the vertices in the connected component of v as explored

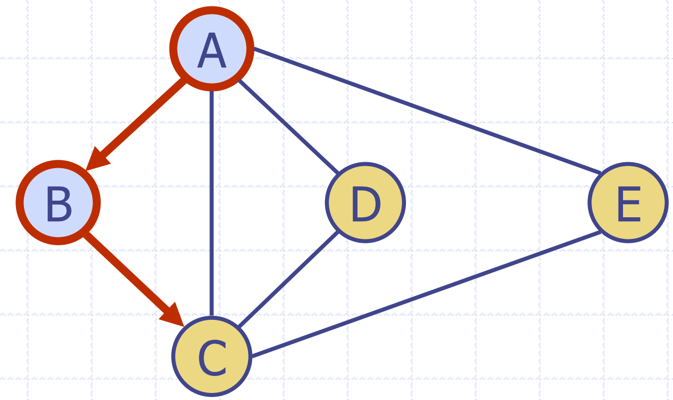
```
Label v as explored
for each edge, e, that is incident to v in G do
    if e is unexplored then
        Let w be the end vertex of e opposite from v
        if w is unexplored then
            Label e as a discovery edge
            DFS(G, w)
        else
            Label e as a back edge
```

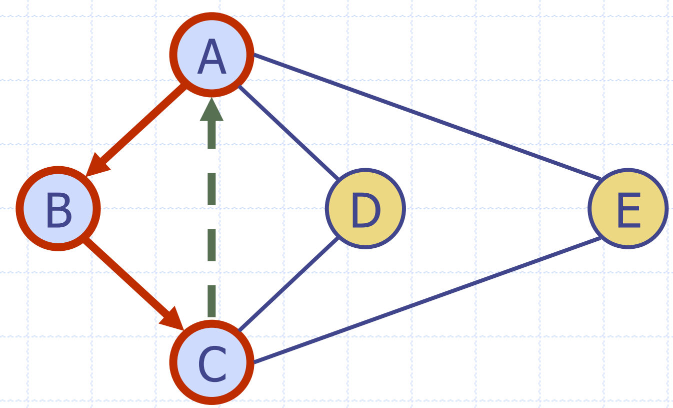
Example



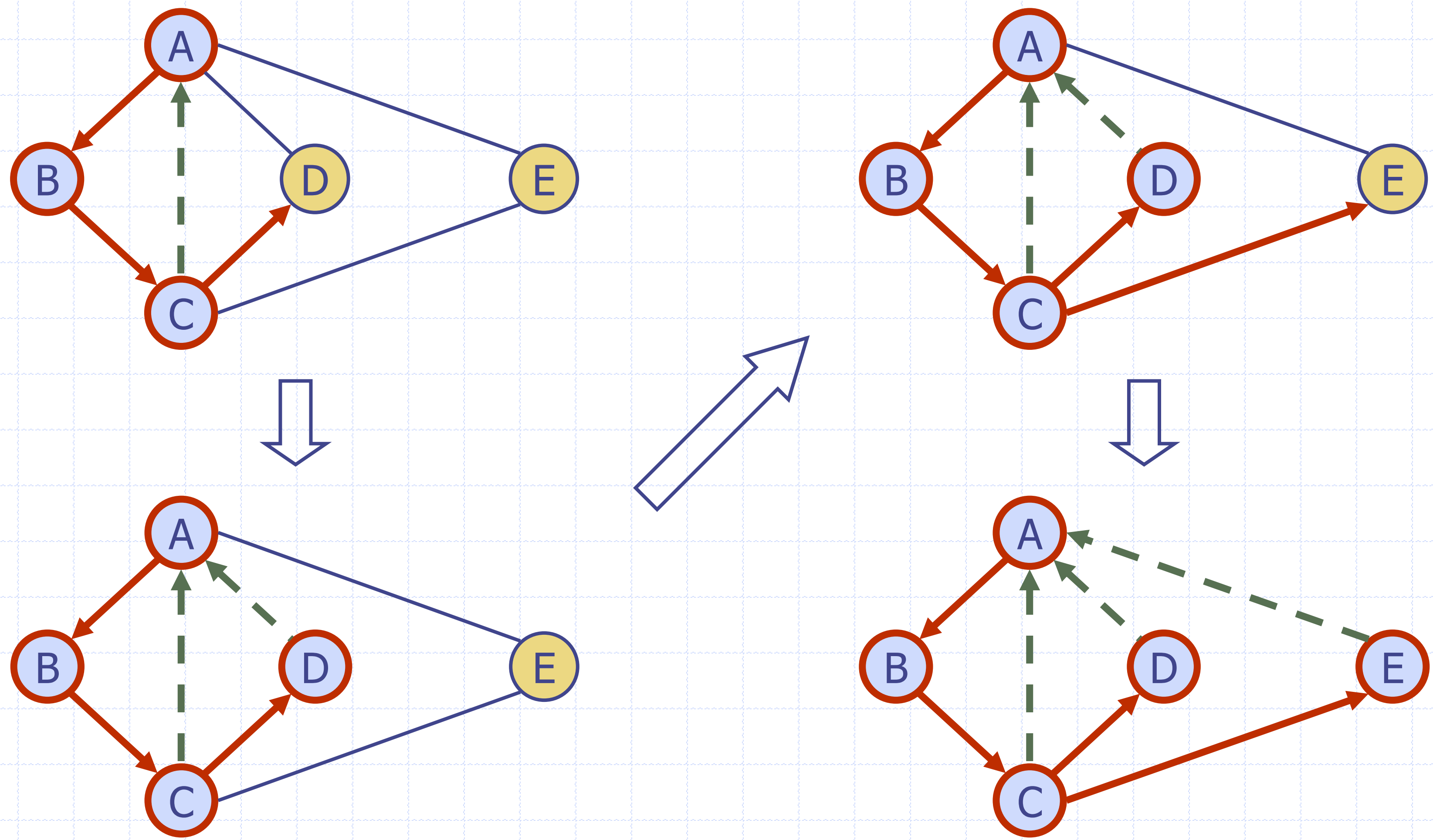
unexplored vertex
visited vertex
unexplored edge
discovery edge
back edge



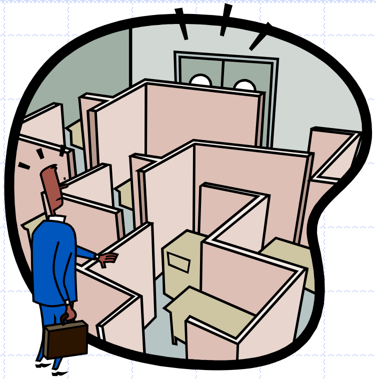




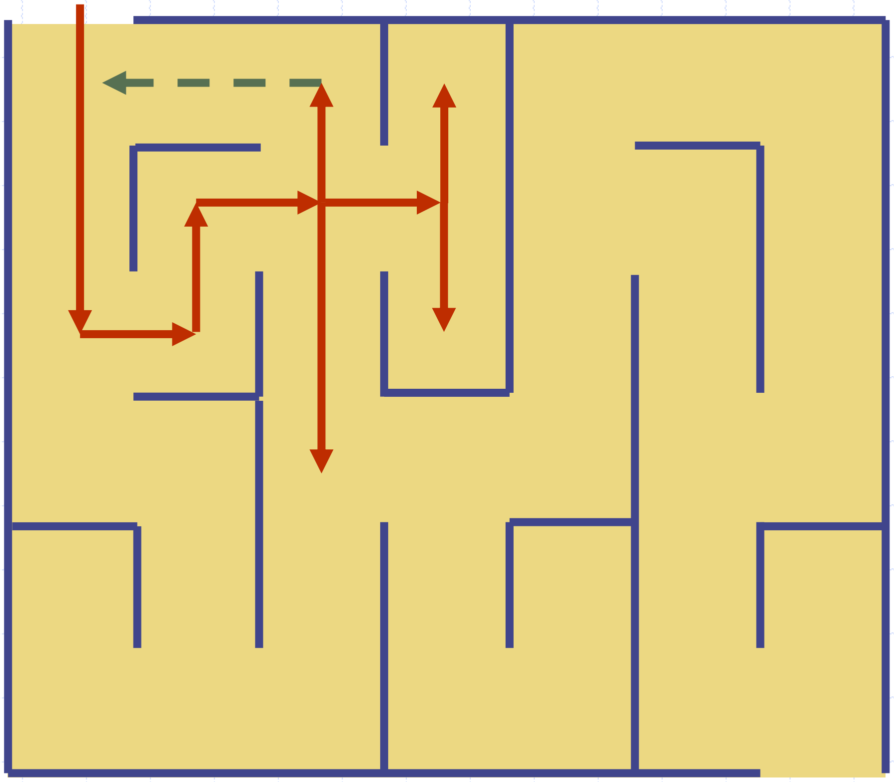
Example (cont.)



DFS and Maze Traversal



- The DFS algorithm is similar to a classic strategy for exploring a maze
  - We mark each intersection, corner and dead end (vertex) visited
  - We mark each corridor (edge) traversed
  - We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack)



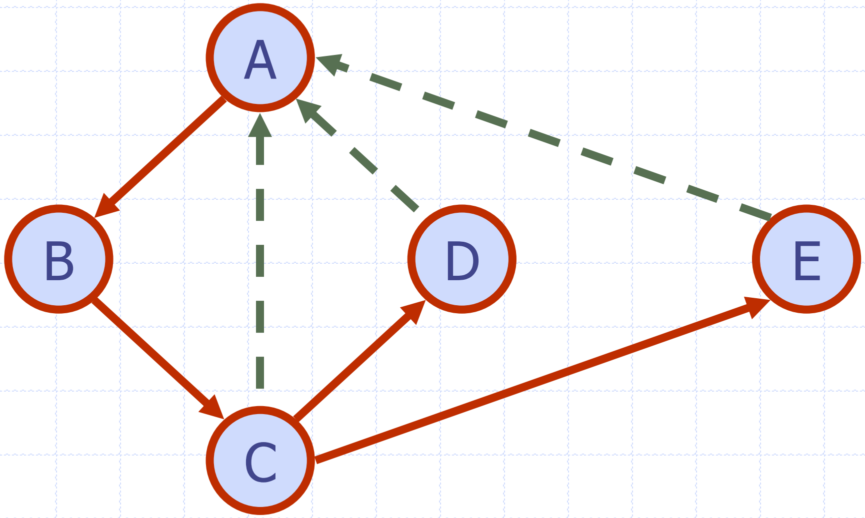
Properties of DFS

Property 1

DFS*(*G*, *v*)** visits all the vertices and edges in the connected component of ***v

Property 2

The discovery edges labeled by ***DFS*(*G*, *v*)** form a spanning tree of the connected component of ***v***



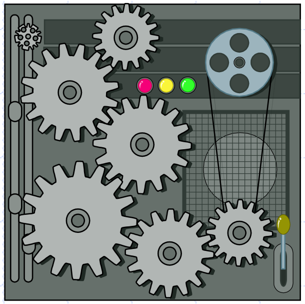
The General DFS Algorithm

- Perform a DFS from each unexplored vertex:

Algorithm DFS(G):
  Input: A graph G
  Output: A labeling of the vertices in each connected component of G as explored

  Initially label each vertex in v as unexplored
  for each vertex, v, in G **do**
    if v is unexplored **then**
      DFS(G, v)

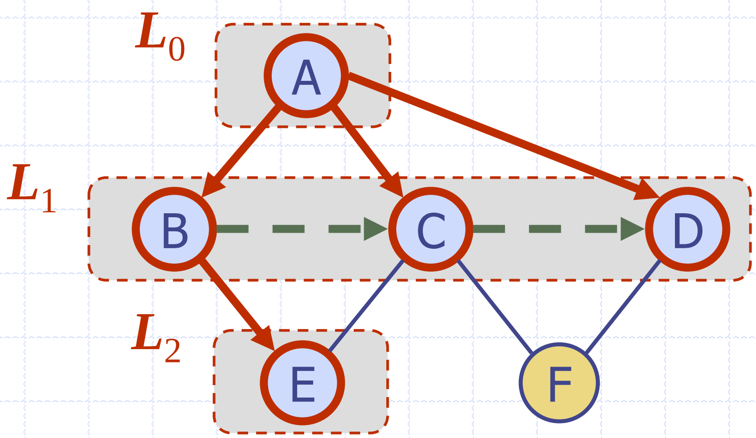
Analysis of DFS



- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
  - once as UNEXPLORED
  - once as VISITED
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or BACK
- Method incidentEdges is called once for each vertex
- DFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
  - Recall that $\Sigma_v \deg(v) = 2m$

Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Breadth-First Search



Breadth-First Search

- Breadth-first search (BFS) is a general technique for traversing a graph
- A BFS traversal of a graph G
  - Visits all the vertices and edges of G
  - Determines whether G is connected
  - Computes the connected components of G
  - Computes a spanning forest of G
- BFS on a graph with n vertices and m edges takes $O(n + m)$ time
- BFS can be further extended to solve other graph problems
  - Find and report a path with the minimum number of edges between two given vertices
  - Find a simple cycle, if there is one

BFS Algorithm

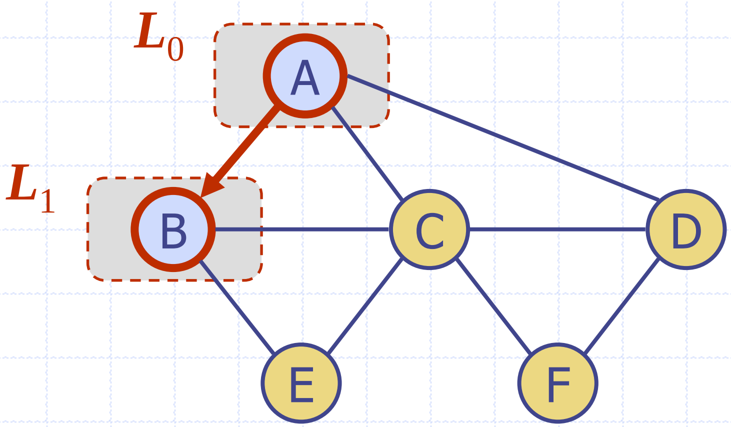
- The algorithm uses "levels" L_i and a mechanism for setting and getting "labels" of vertices and edges.

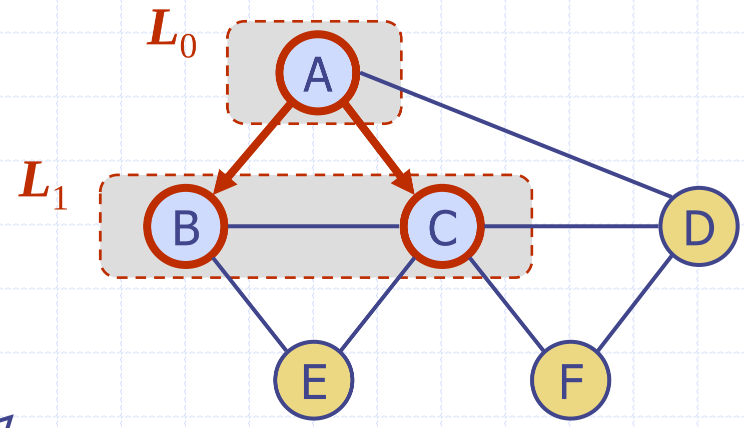
Algorithm BFS(G, s):
  Input: A graph G and a vertex s of G
  Output: A labeling of the edges in the connected component of s as discovery edges and cross edges

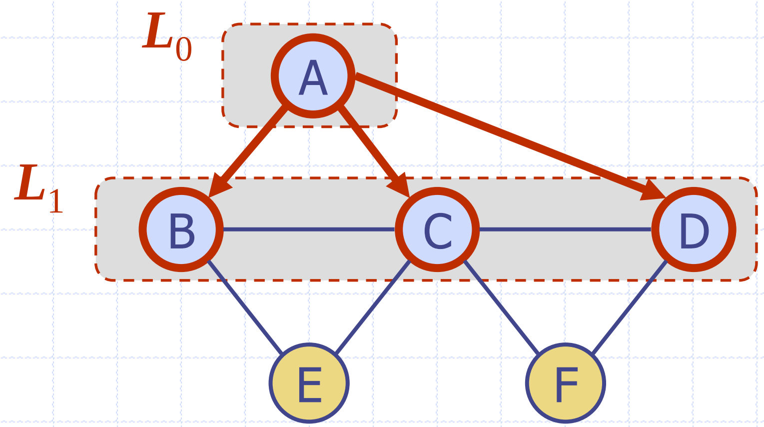
```
Create an empty list, L_0
Mark s as explored and insert s into L_0
i ← 0
while L_i is not empty do
    create an empty list, L_{i+1}
    for each vertex, v, in L_i do
        for each edge, e = (v, w), incident on v in G do
            if edge e is unexplored then
                if vertex w is unexplored then
                    Label e as a discovery edge
                    Mark w as explored and insert w into L_{i+1}
                else
                    Label e as a cross edge
    i ← i + 1
```

Example

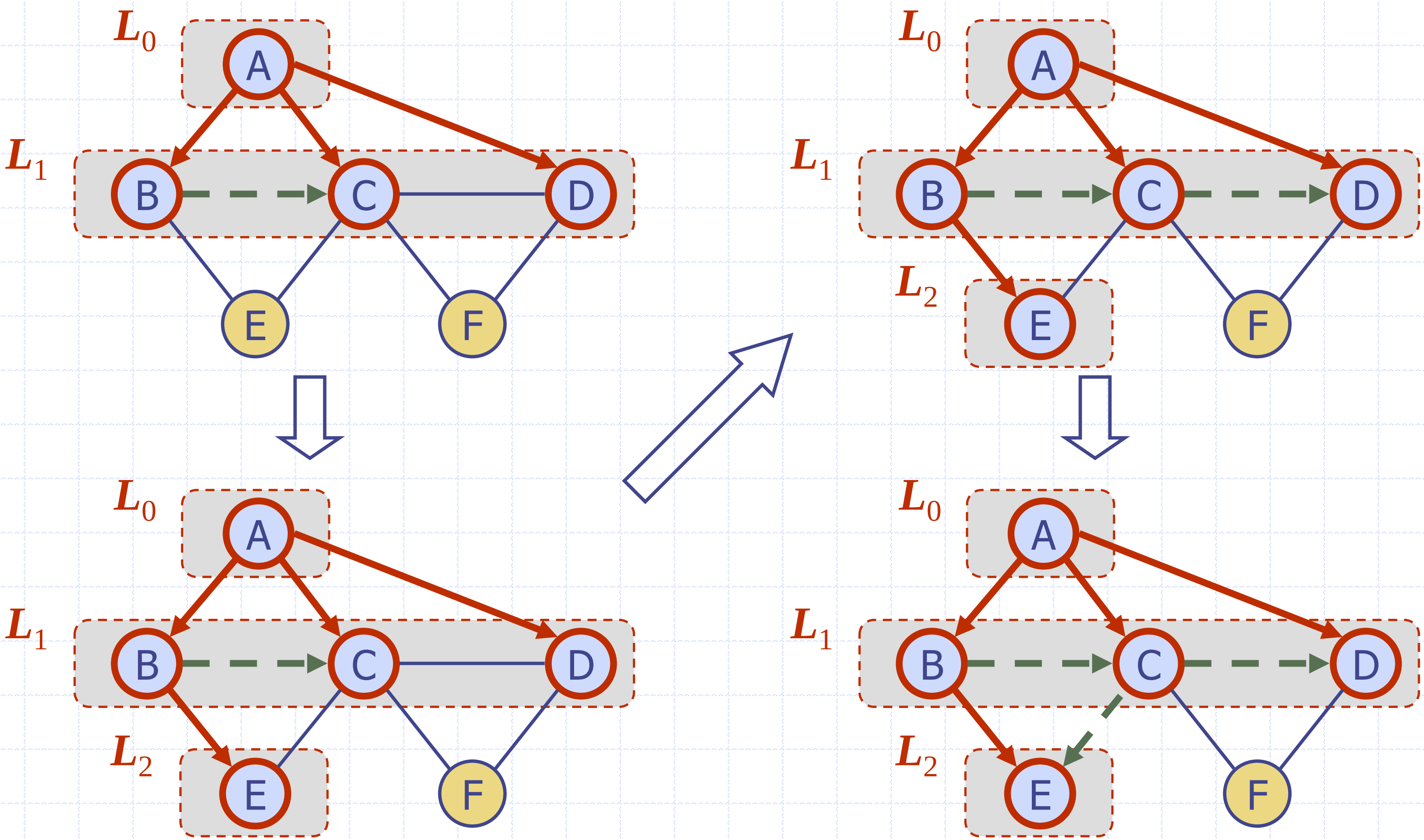
- A unexplored vertex
- A visited vertex
- unexplored edge
- discovery edge
- cross edge



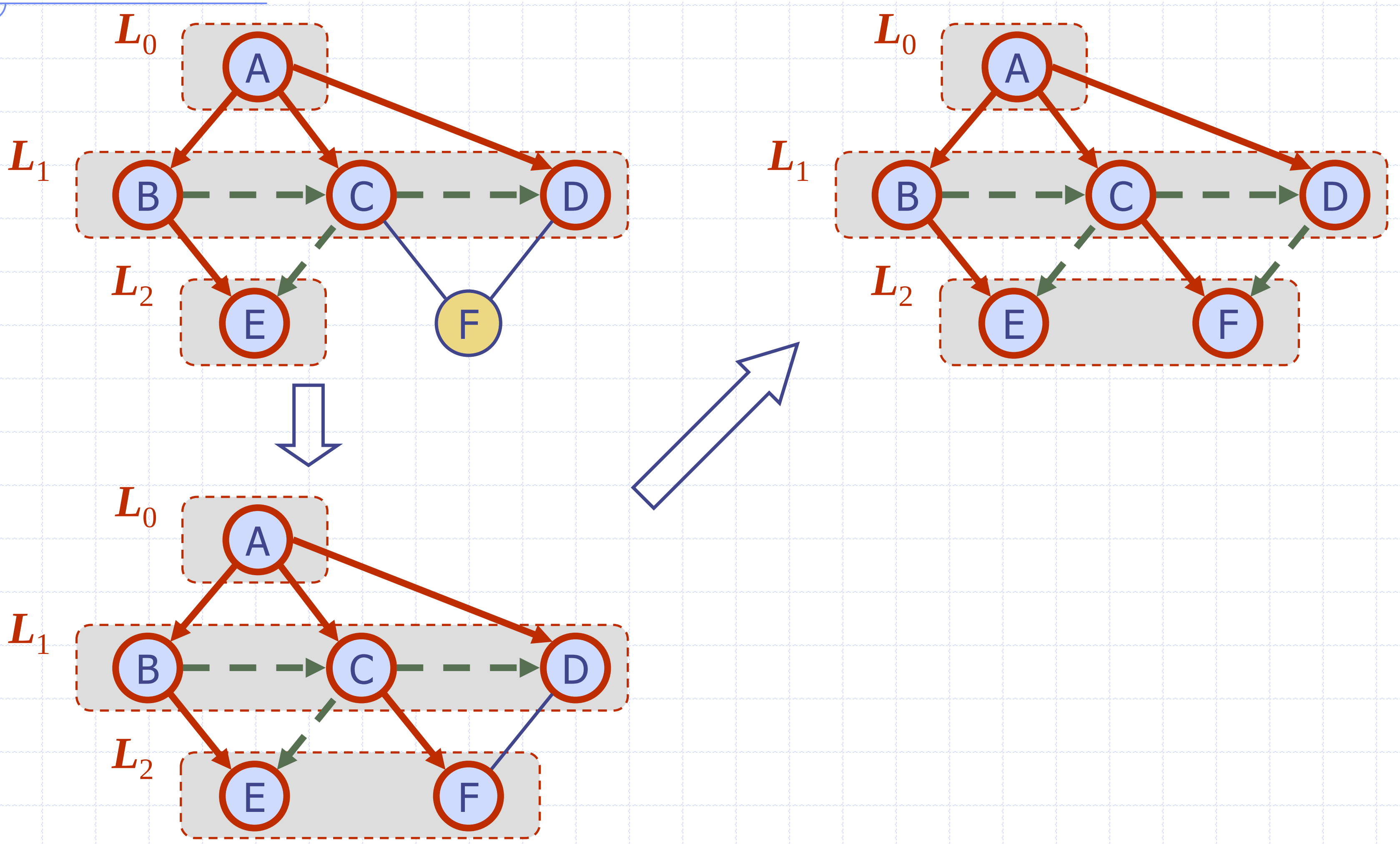




Example (cont.)



Example (cont.)



Properties

Notation

G_s: connected component of s

Property 1

$BFS(G, s)$ visits all the vertices and edges of G_s

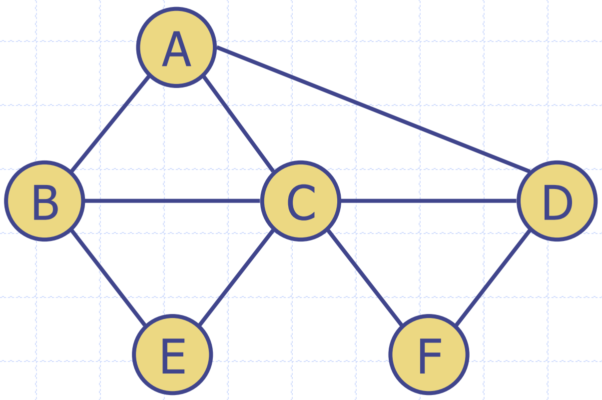
Property 2

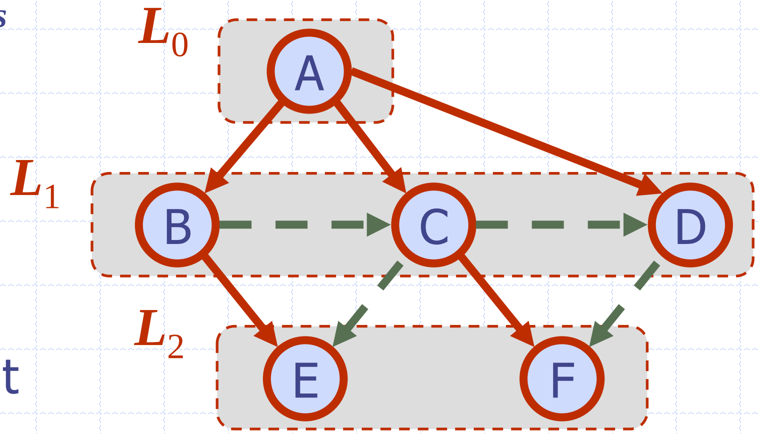
The discovery edges labeled by $BFS(G, s)$ form a spanning tree T_s of G_s

Property 3

For each vertex v in L_i

- The path of T_s from s to v has i edges
- Every path from s to v in G_s has at least i edges





Analysis

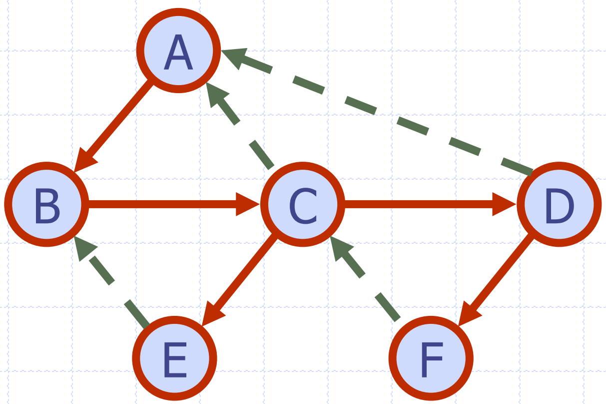
- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
  - once as UNEXPLORED
  - once as VISITED
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or CROSS
- Each vertex is inserted once into a sequence L_i
- Method incidentEdges is called once for each vertex
- BFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
  - Recall that $\Sigma_v \deg(v) = 2m$

Applications

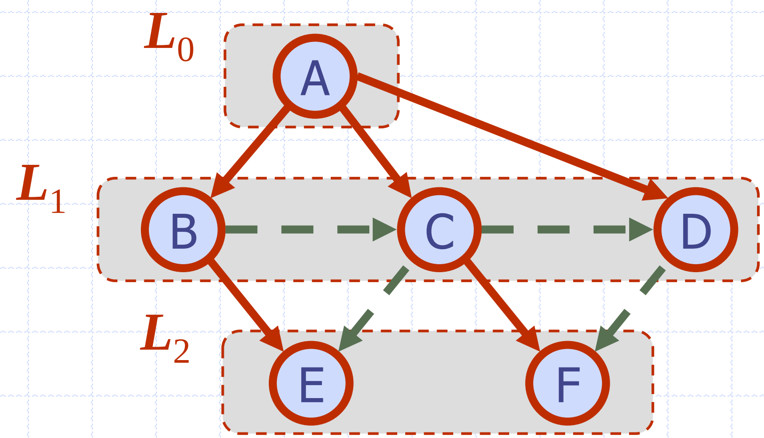
- We can use the BFS traversal algorithm, for a graph G, to solve the following problems in $O(n + m)$ time
  - Compute the connected components of G
  - Compute a spanning forest of G
  - Find a simple cycle in G, or report that G is a forest
  - Given two vertices of G, find a path in G between them with the minimum number of edges, or report that no such path exists

DFS vs. BFS

Applications	DFS	BFS
Spanning forest, connected components, paths, cycles	√	√
Shortest paths		√
Biconnected components	√	



DFS

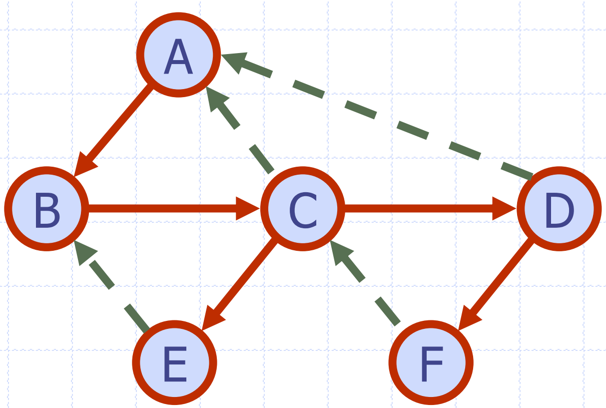


BFS

DFS vs. BFS (cont.)

Back edge (v,w)

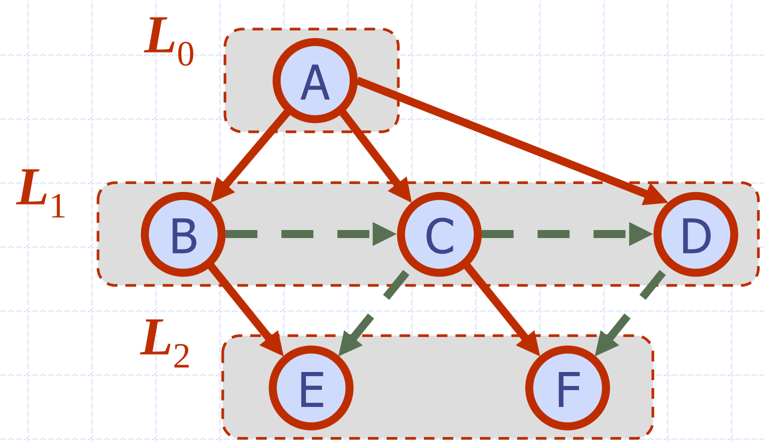
- w is an ancestor of v in the tree of discovery edges



DFS

Cross edge (v,w)

- w is in the same level as v or in the next level



BFS

Cycle detection

- Graph G has a cycle iff DFS **has a back edge**

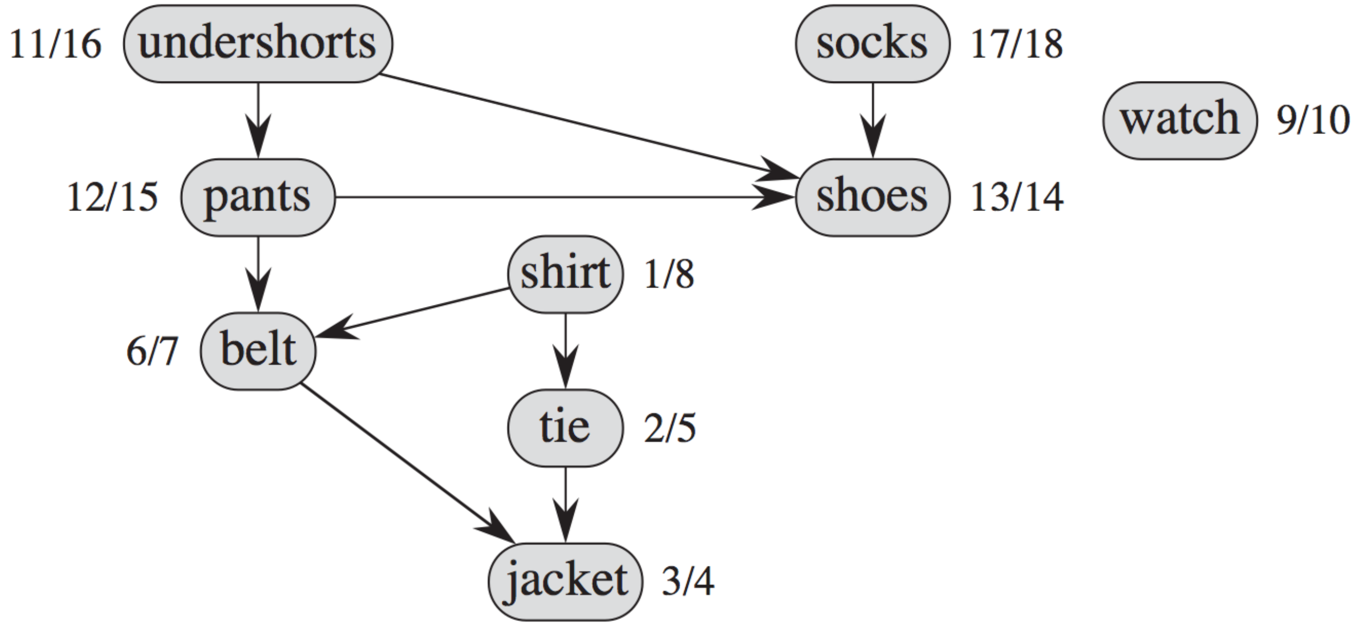
Directed Acyclic Graph = DAG

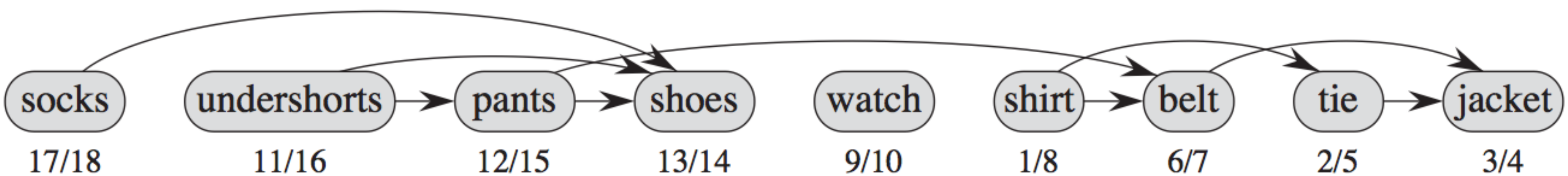
Topological sort

Topological sort of a DAG $G=(V,E)$

1. Run DFS(G), compute finishing times of nodes
2. Output the nodes in decreasing order of finishing times

The Graph – relationship between clothing procedures

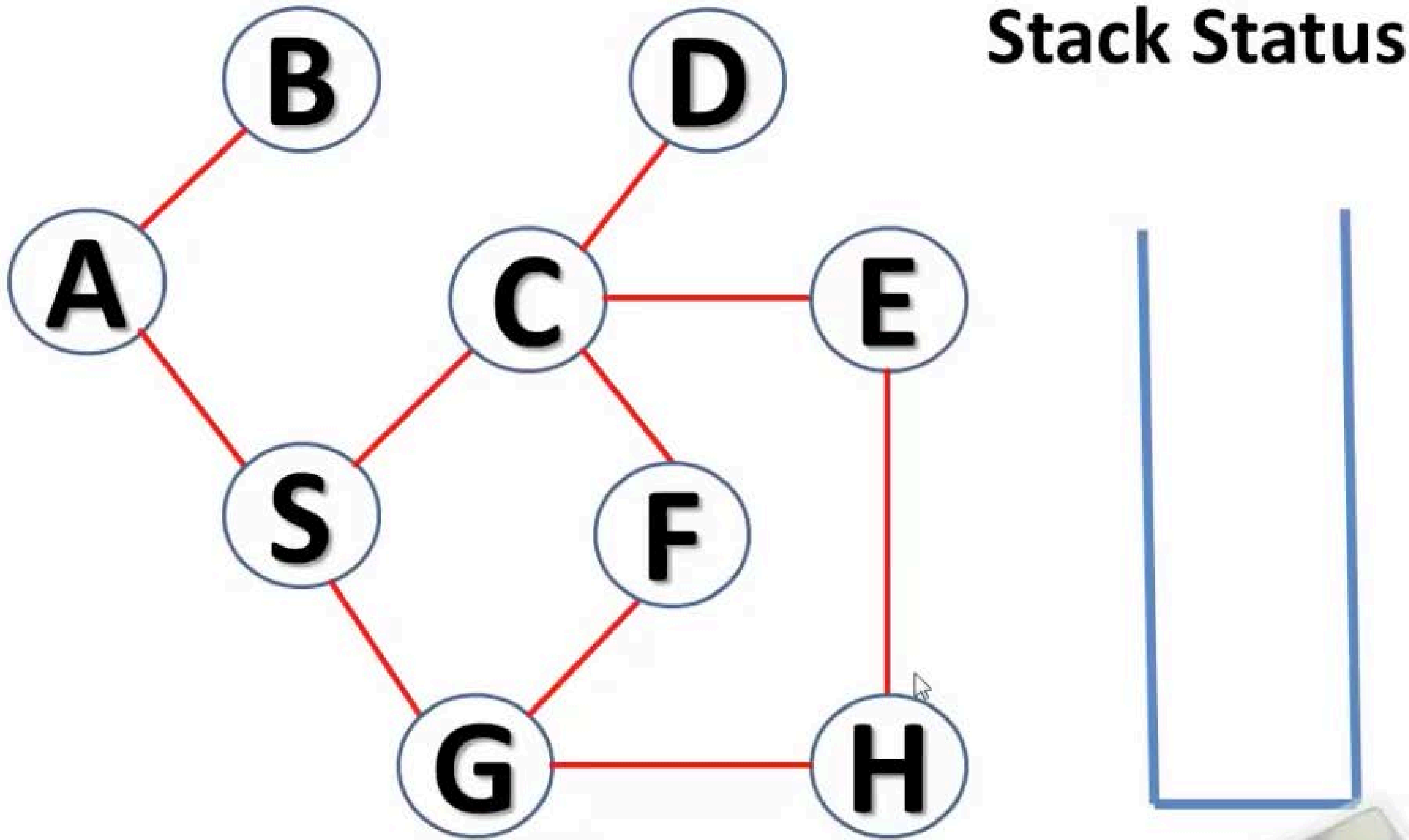




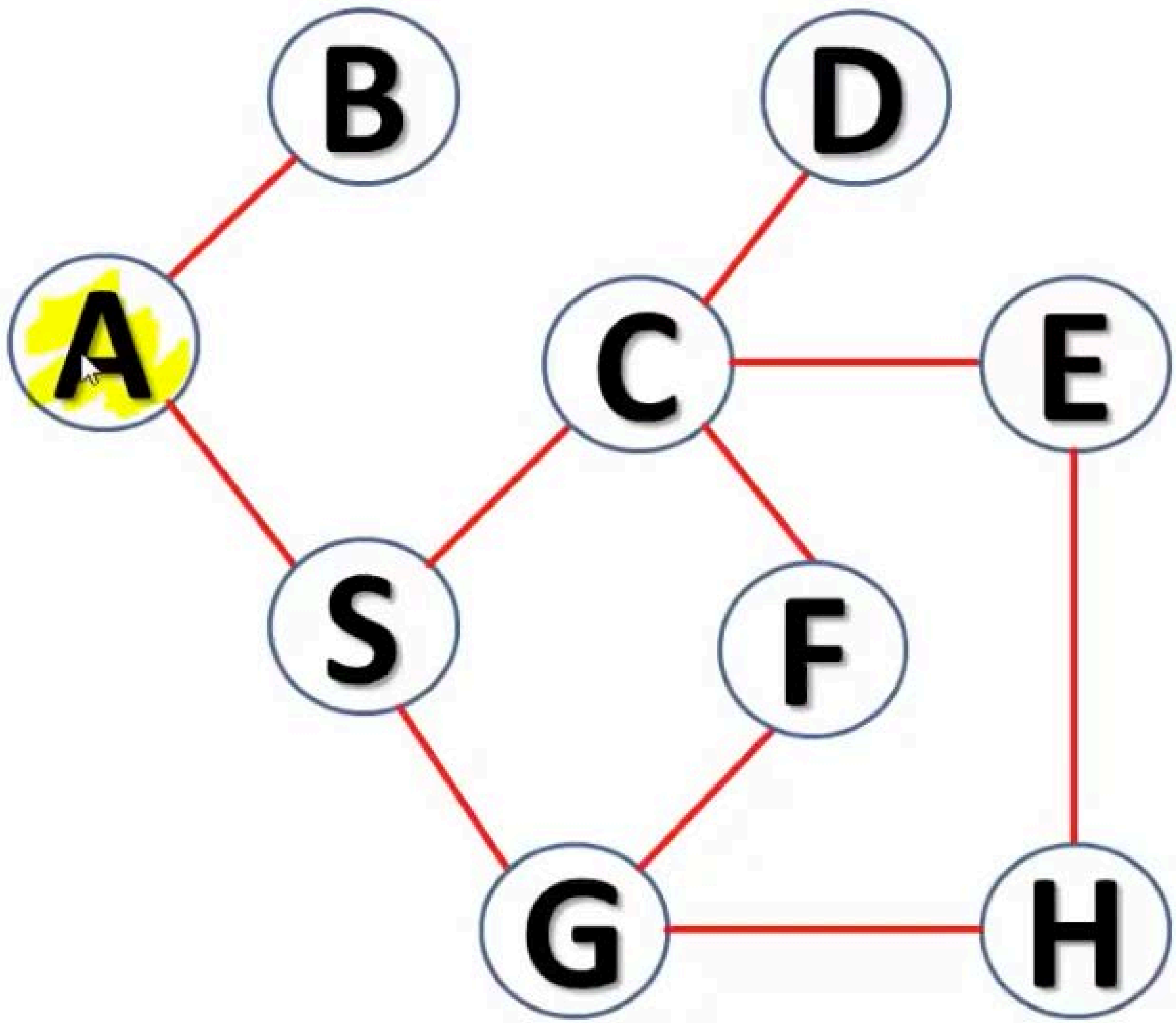
The Topological sort – a workable sequence of clothing

1. DFS WITH STACK

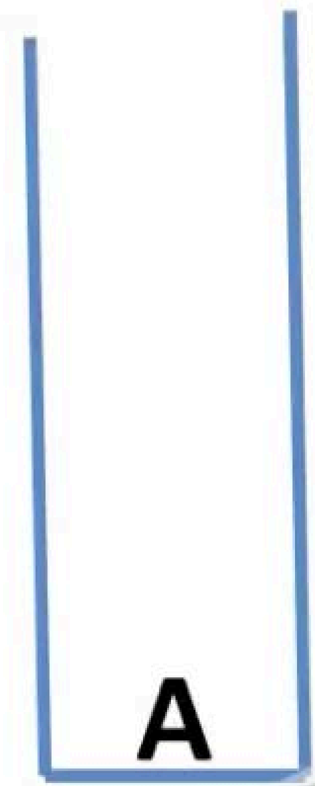
DEPTH FIRST SEARCH



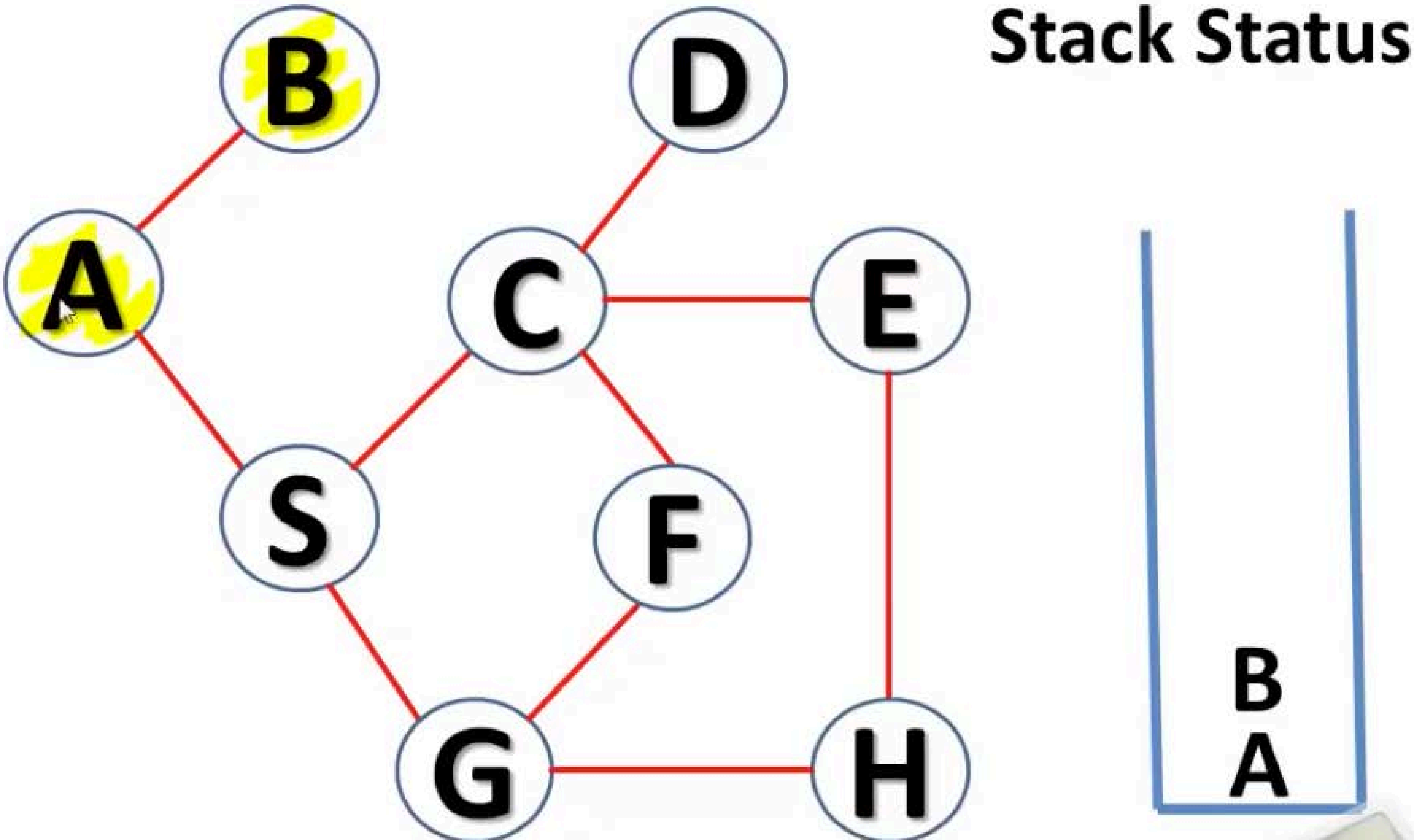
DEPTH FIRST SEARCH



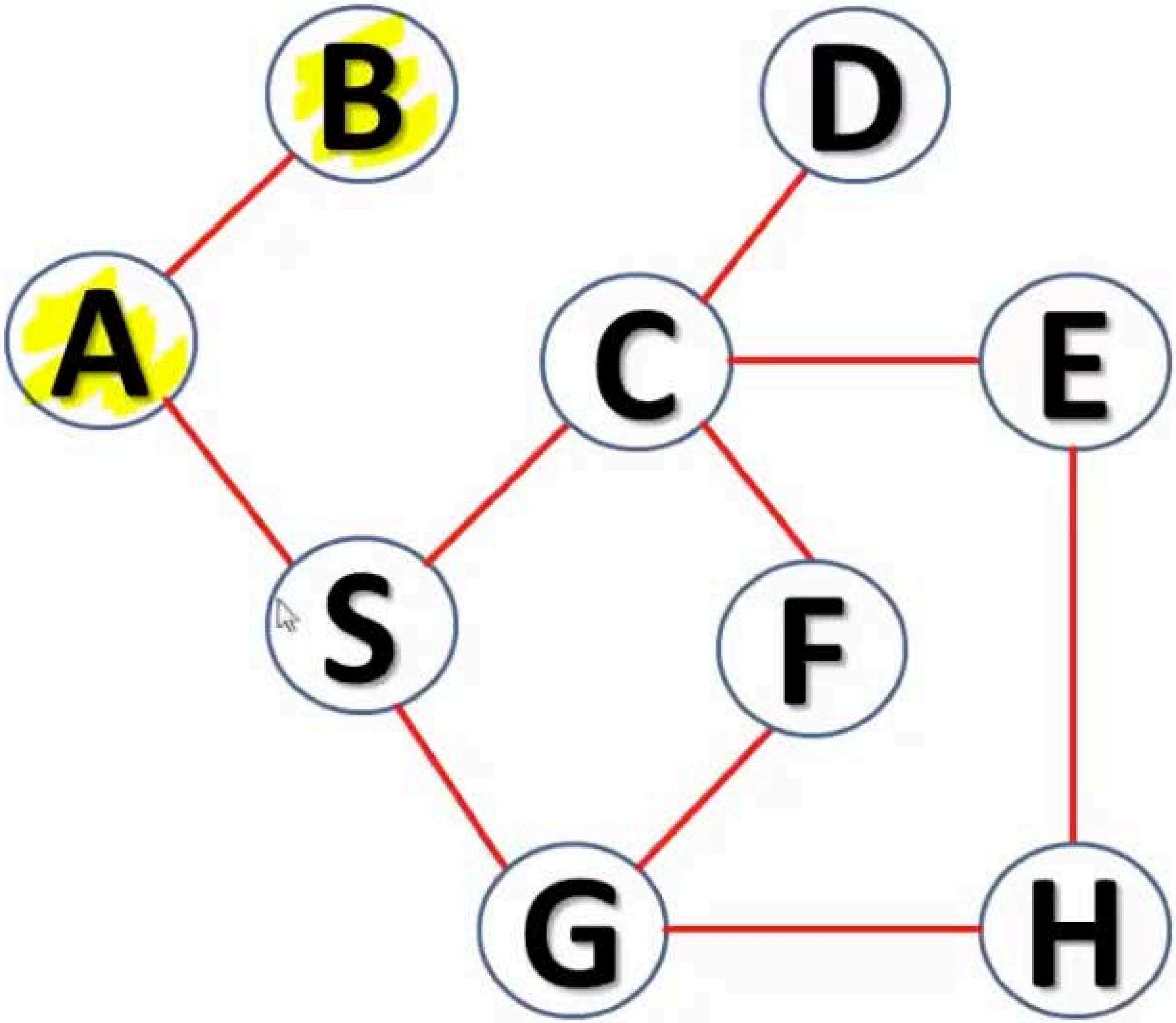
Stack Status



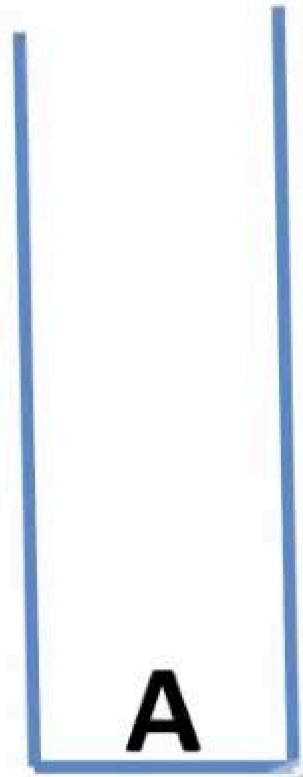
DEPTH FIRST SEARCH



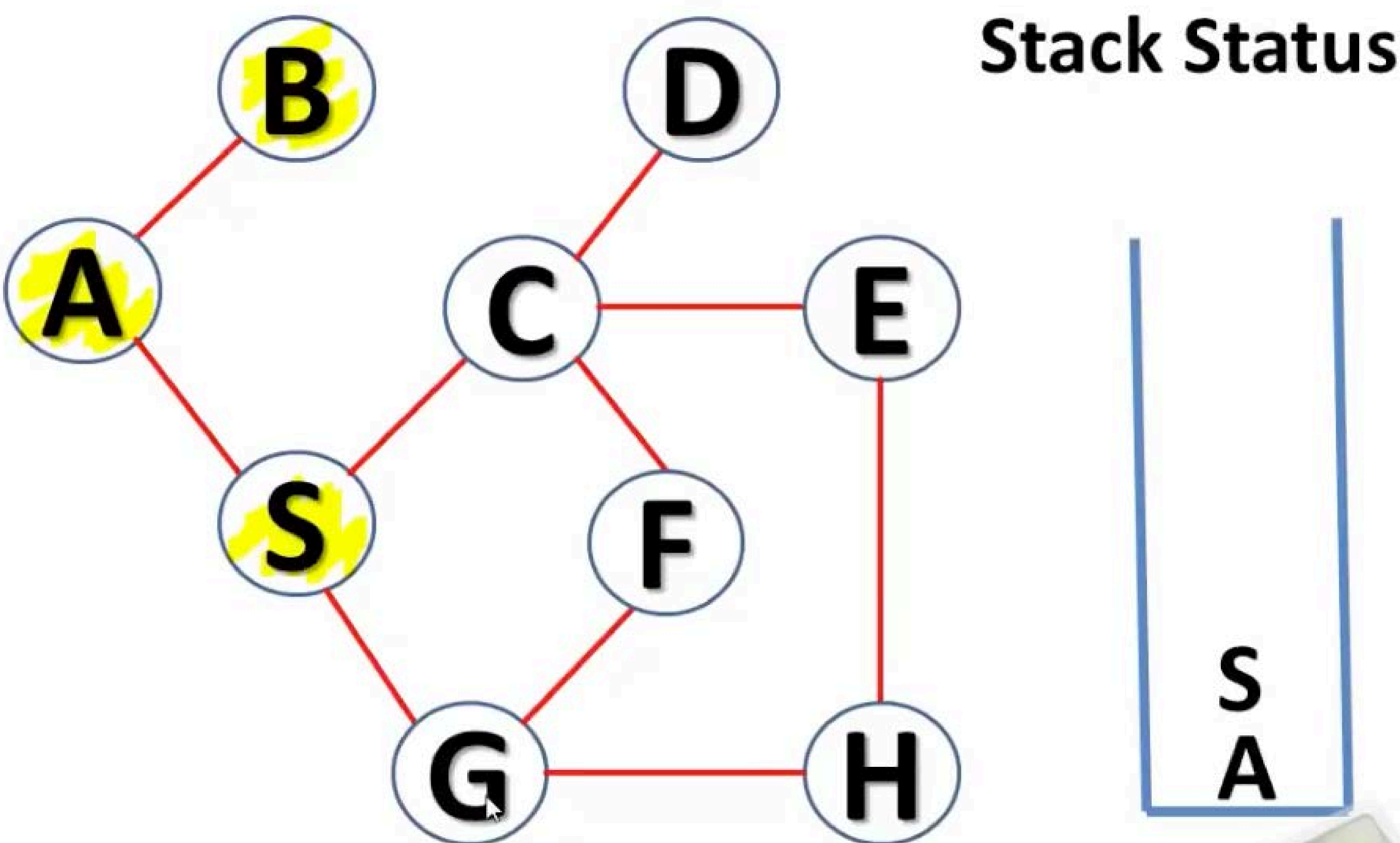
DEPTH FIRST SEARCH



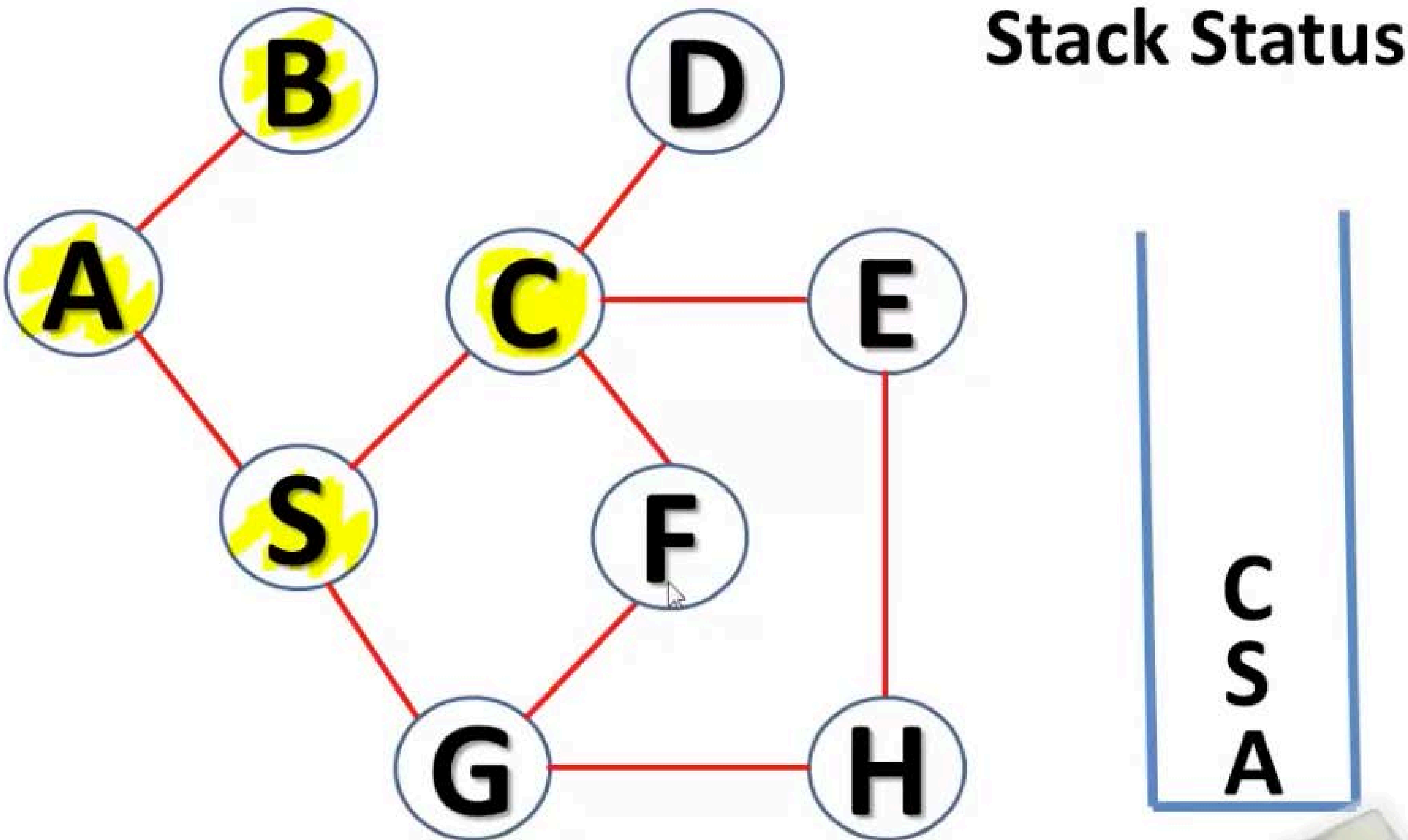
Stack Status



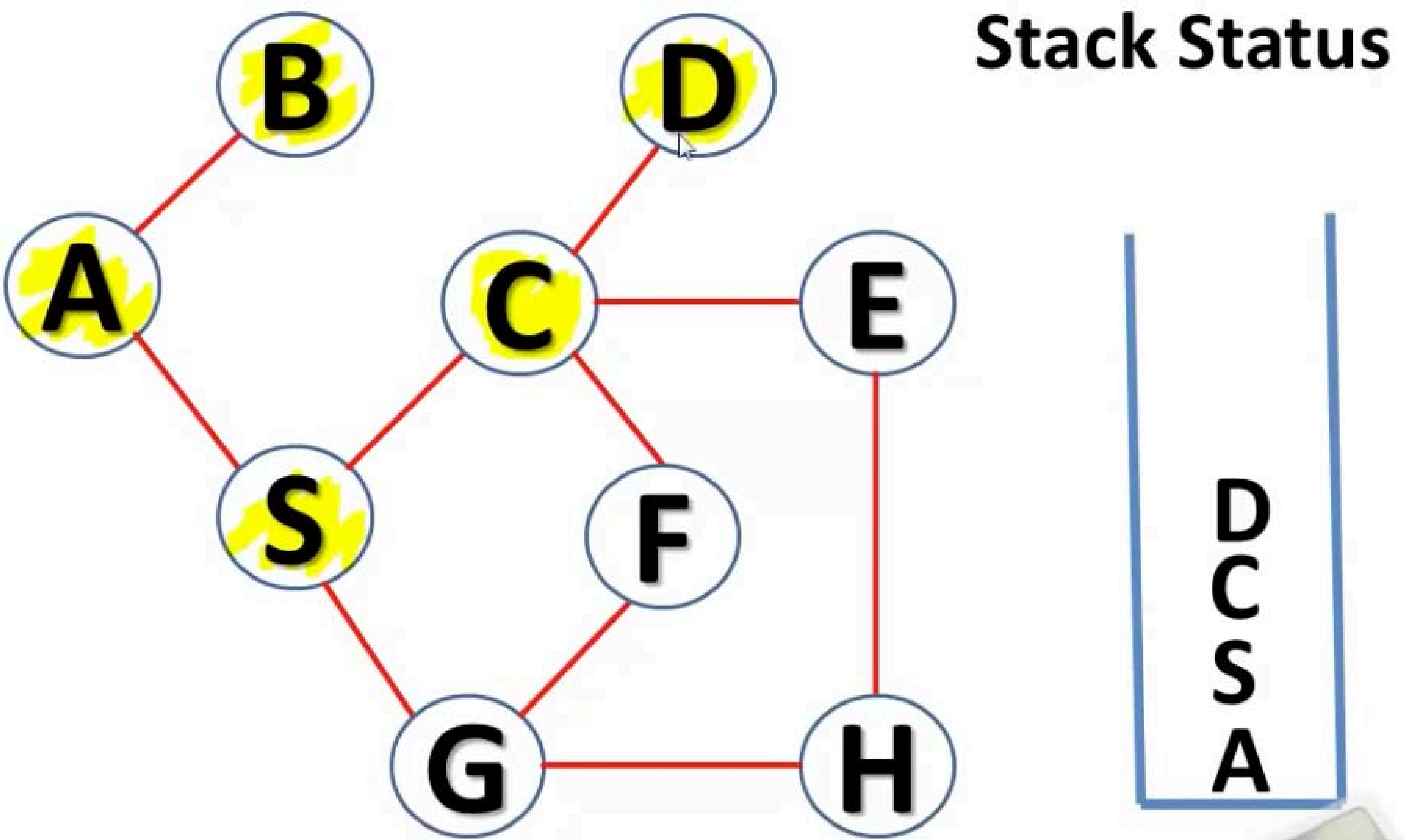
DEPTH FIRST SEARCH



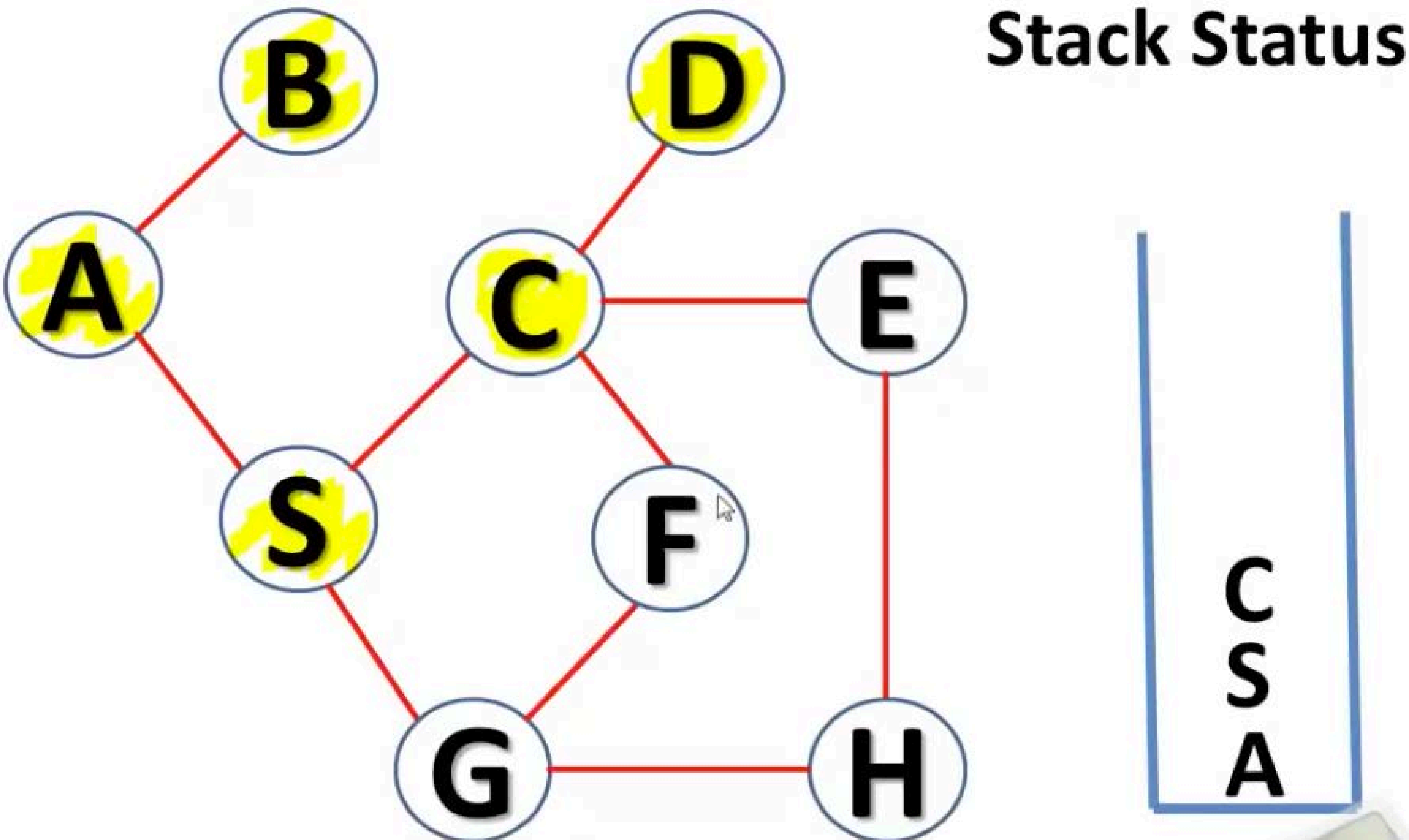
DEPTH FIRST SEARCH



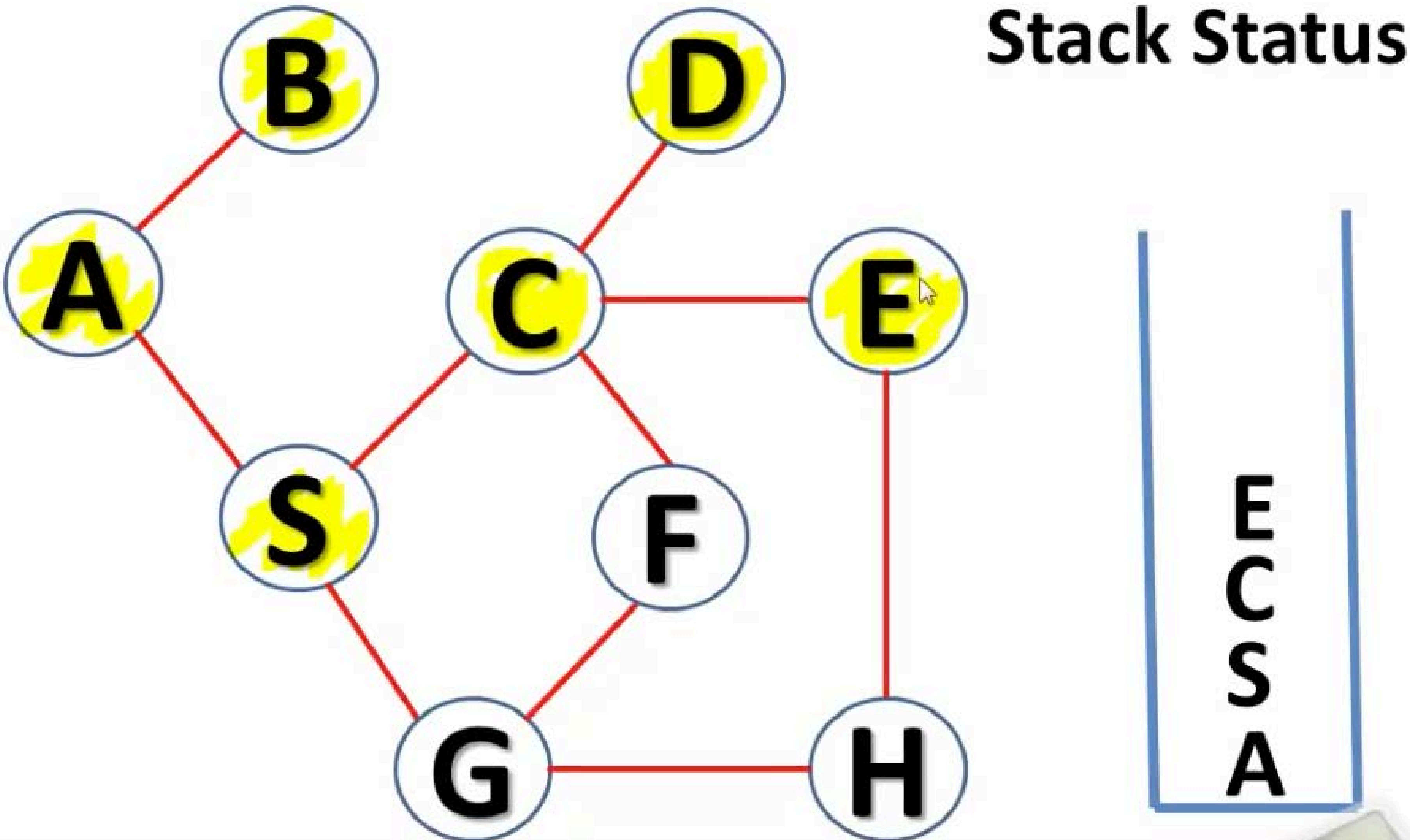
DEPTH FIRST SEARCH



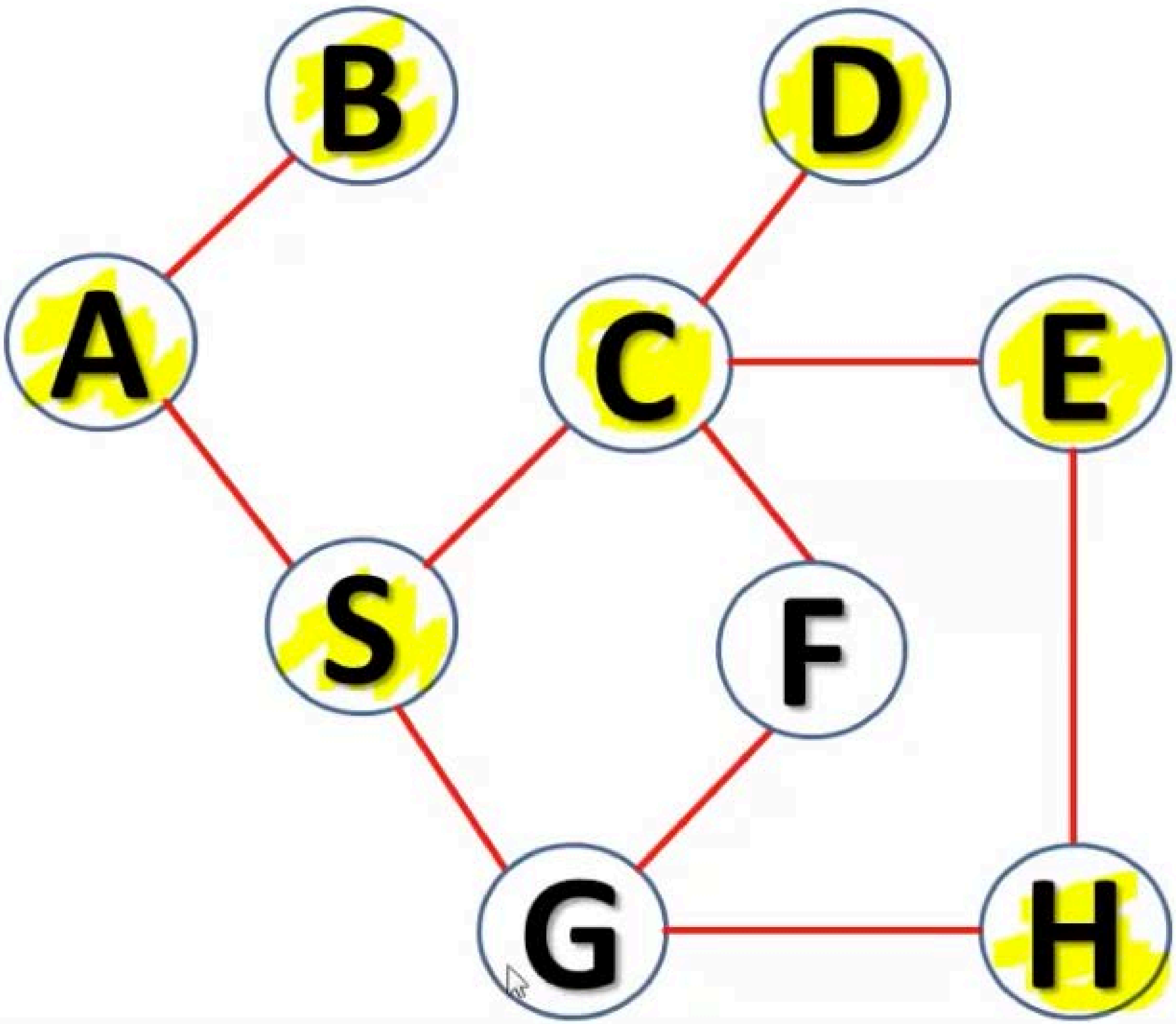
DEPTH FIRST SEARCH



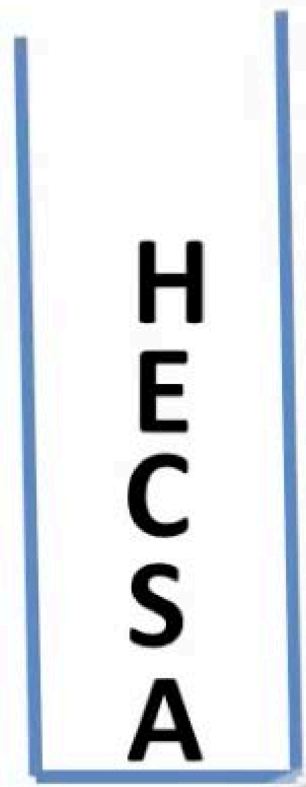
DEPTH FIRST SEARCH



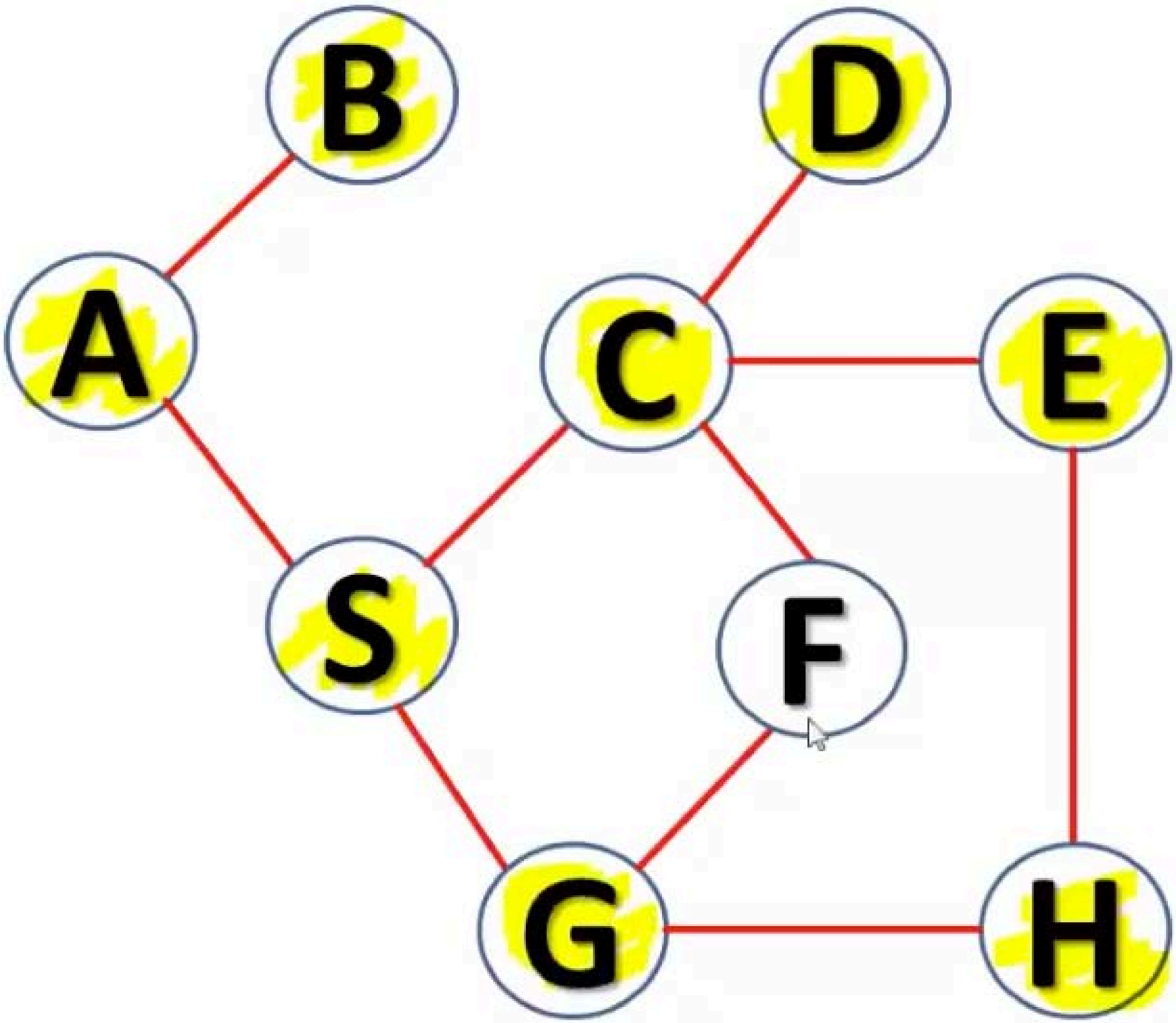
DEPTH FIRST SEARCH



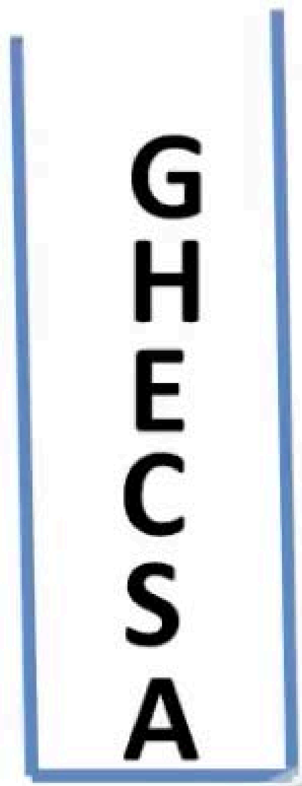
Stack Status



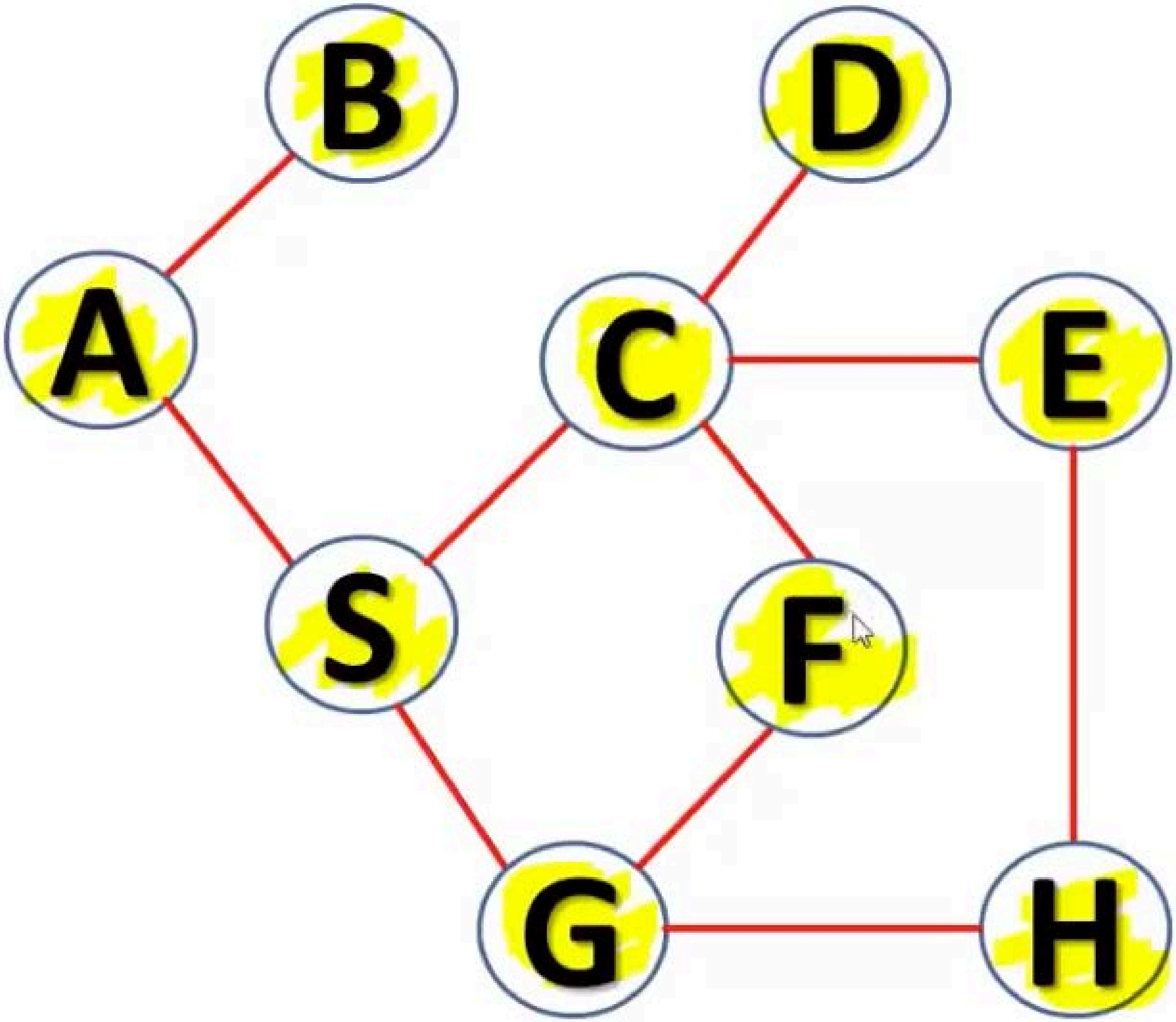
DEPTH FIRST SEARCH



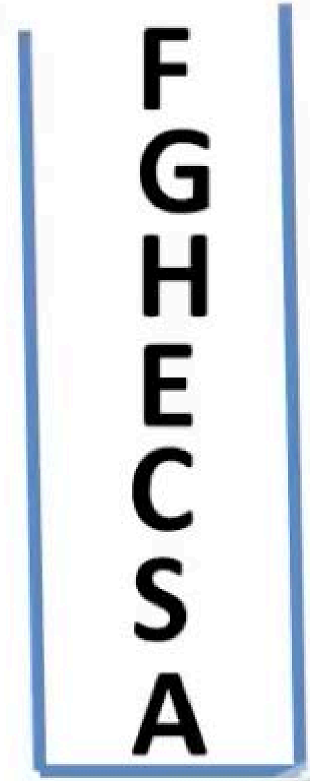
Stack Status



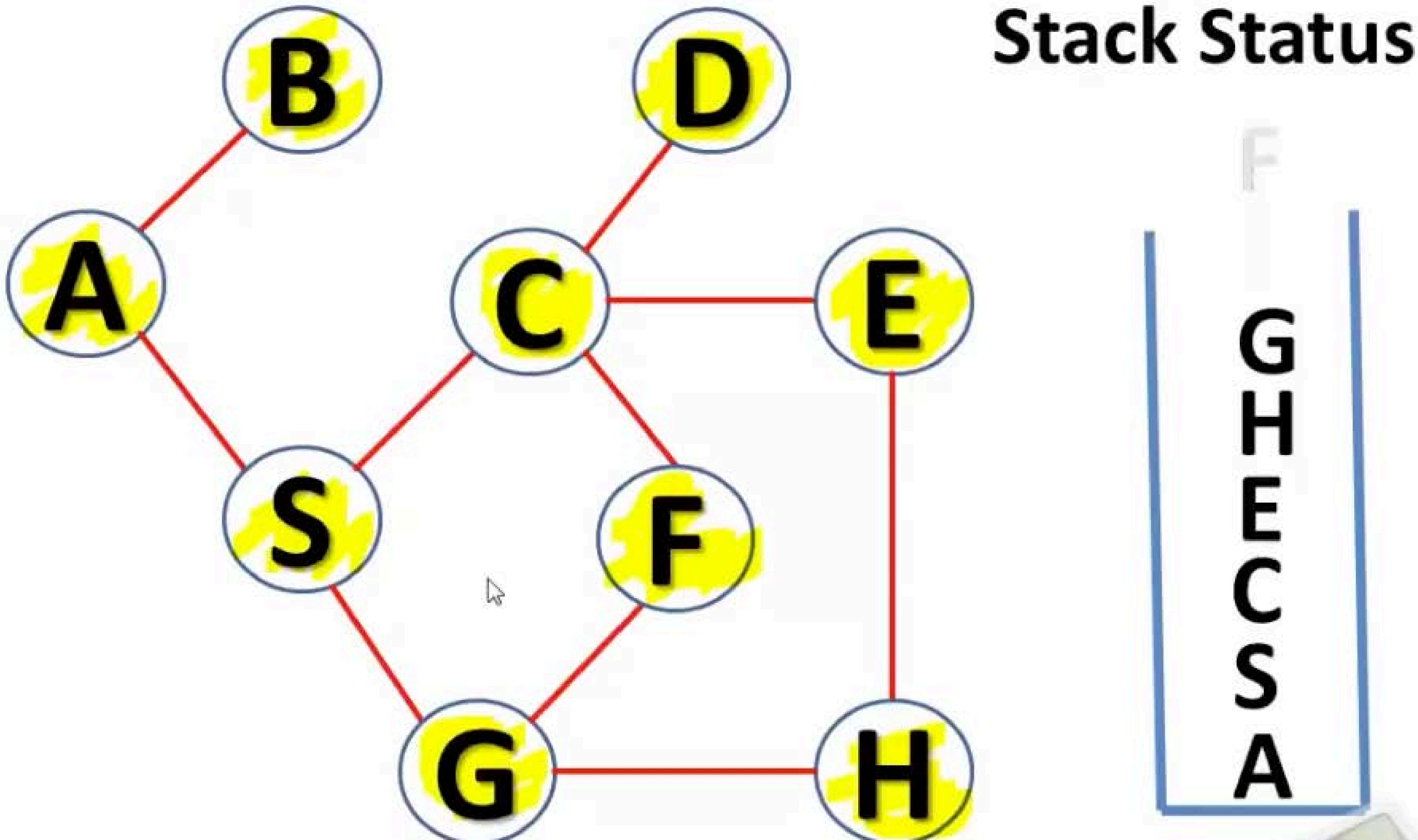
DEPTH FIRST SEARCH



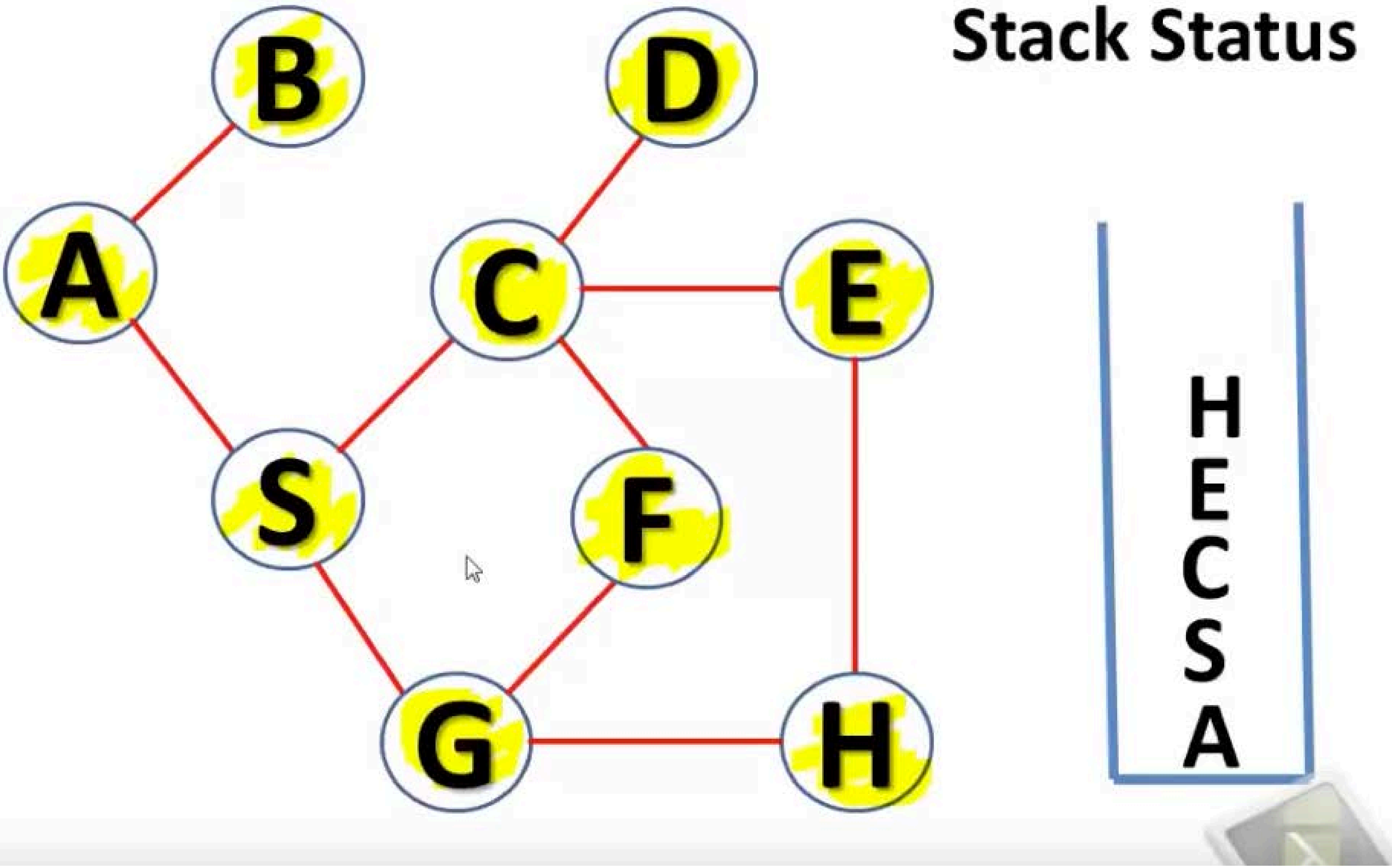
Stack Status



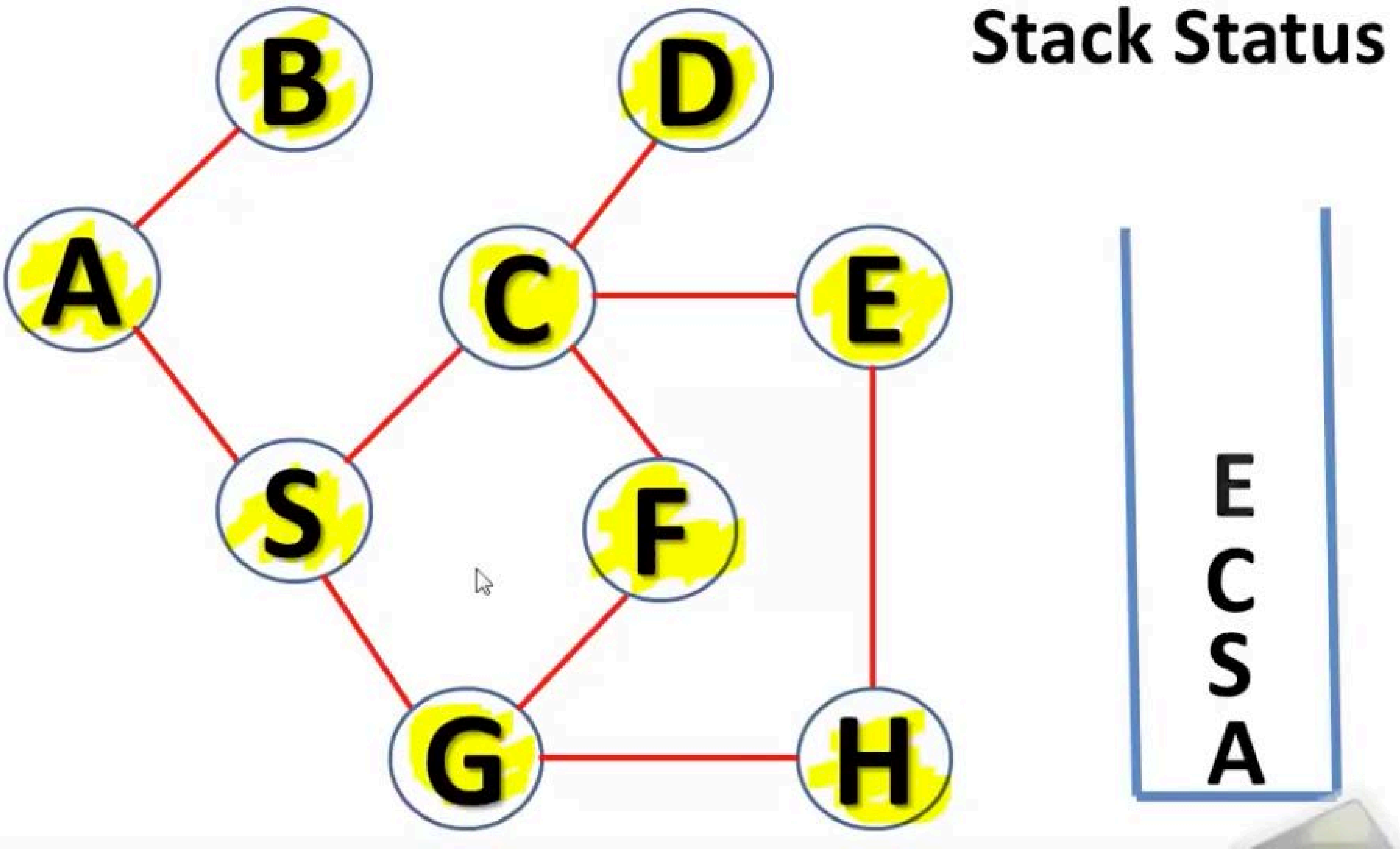
DEPTH FIRST SEARCH



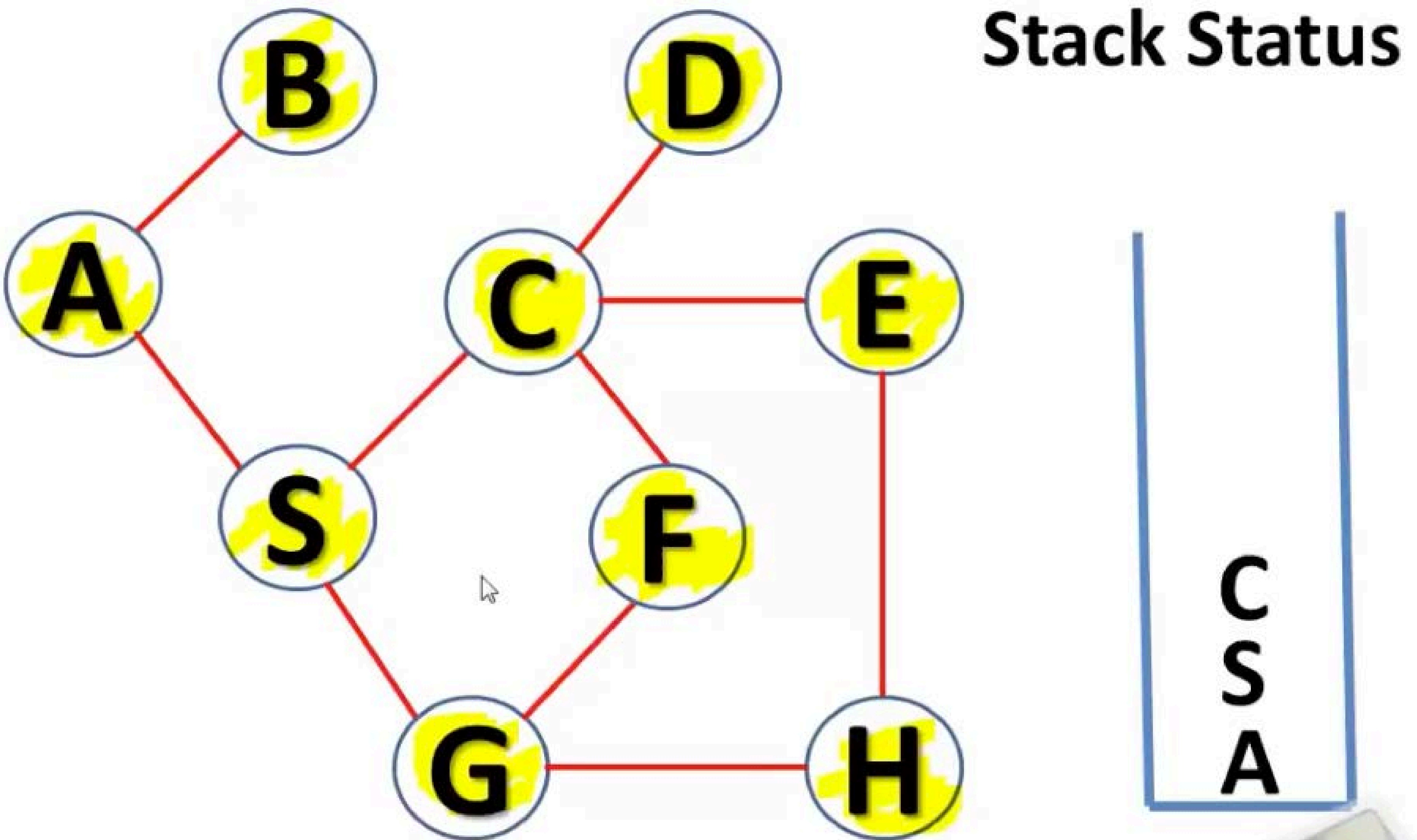
DEPTH FIRST SEARCH



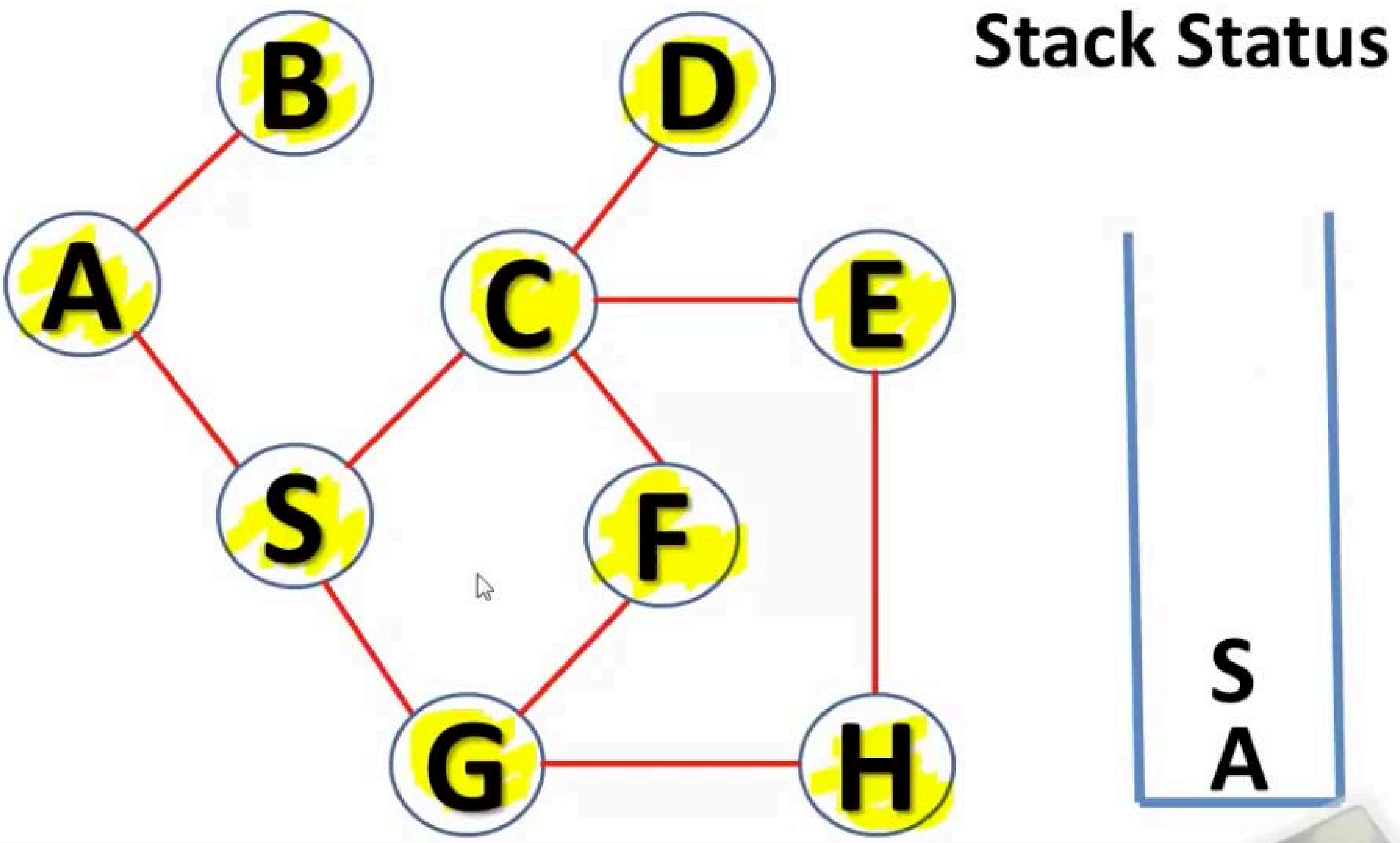
DEPTH FIRST SEARCH



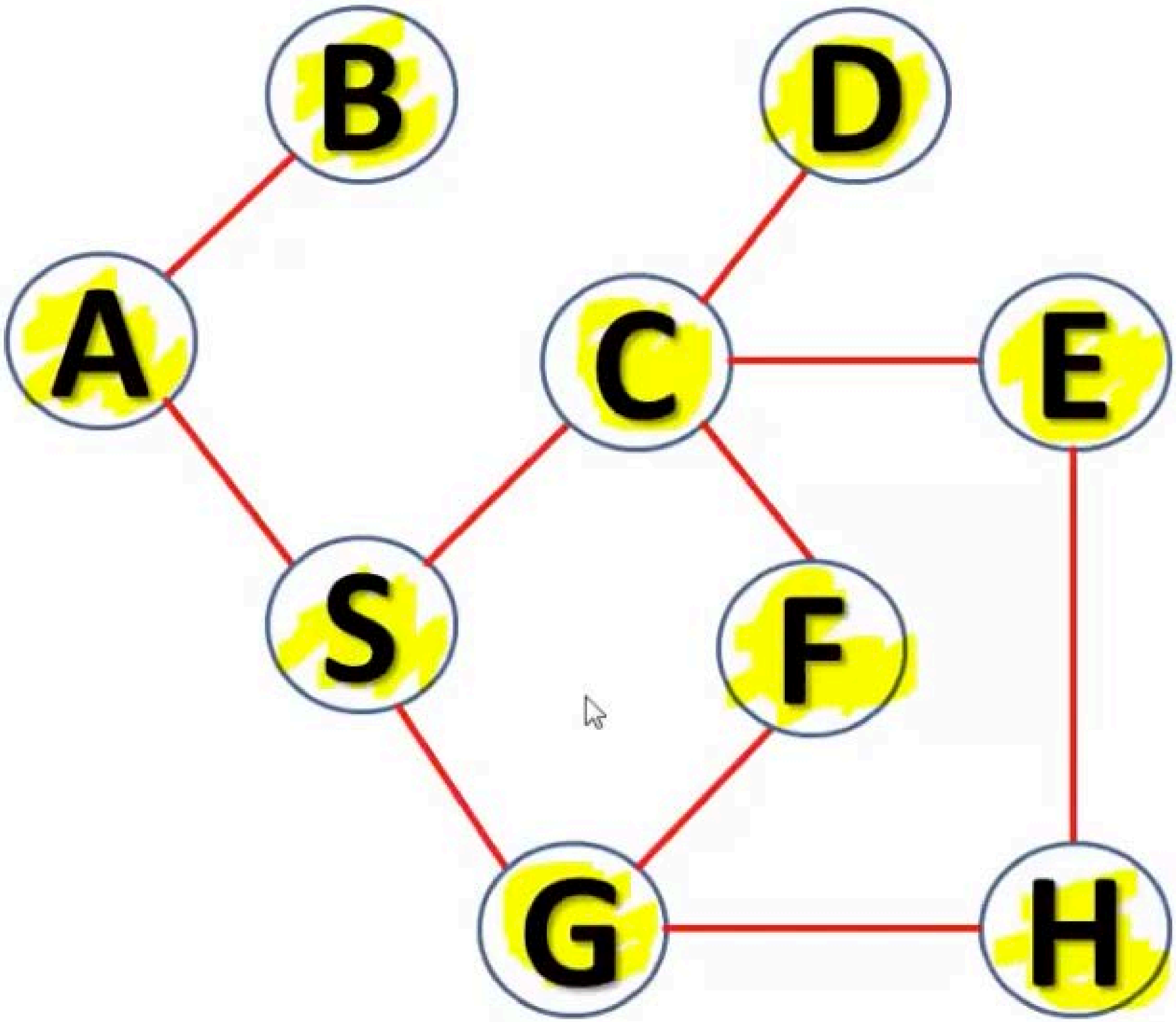
DEPTH FIRST SEARCH



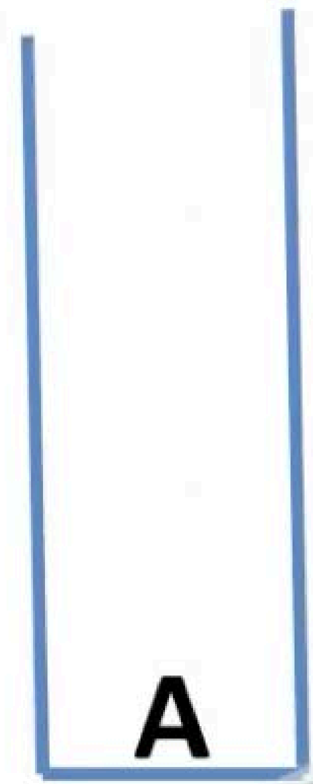
DEPTH FIRST SEARCH



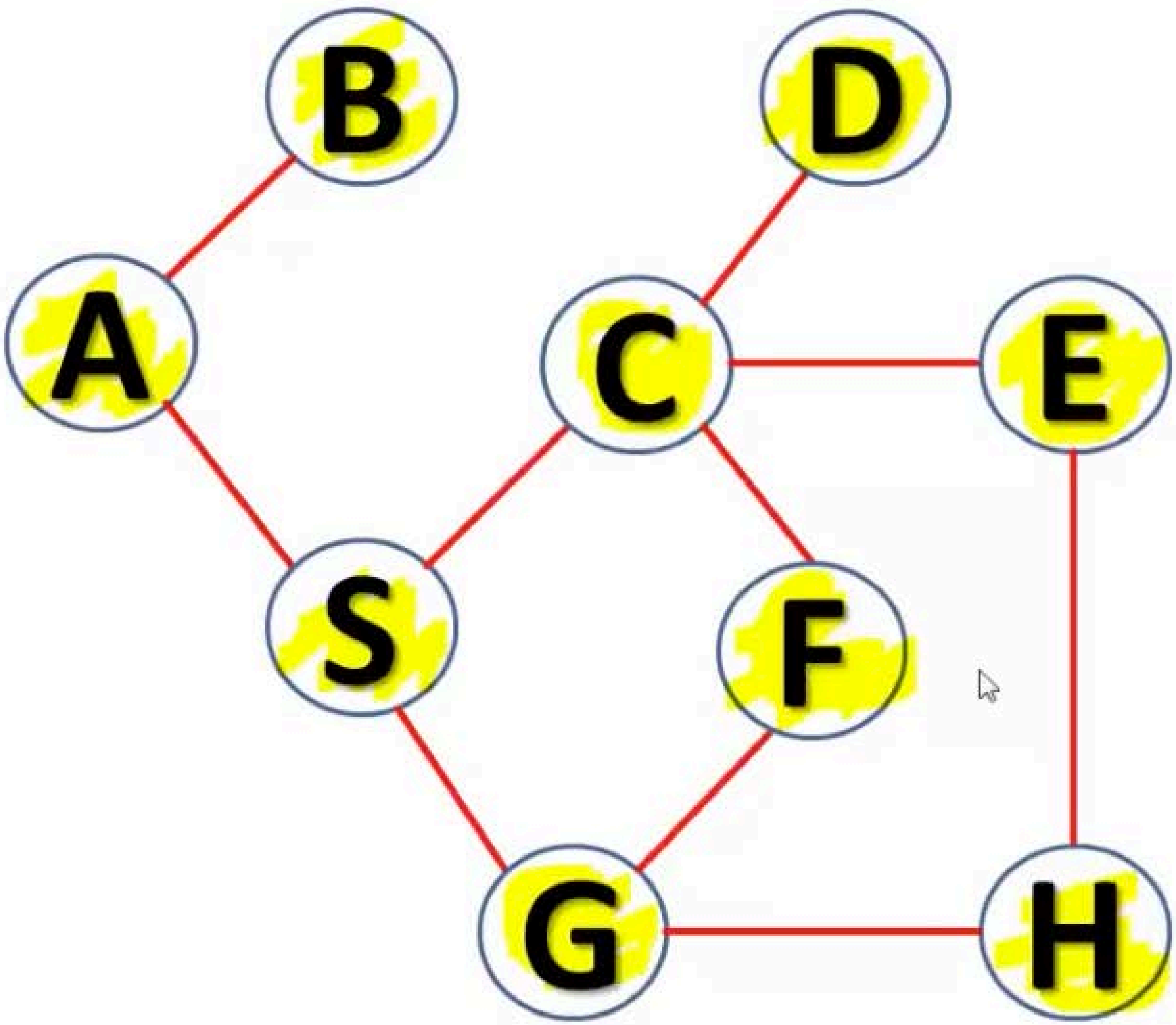
DEPTH FIRST SEARCH



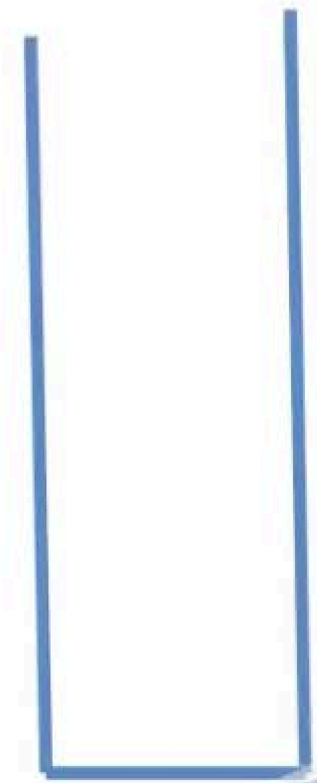
Stack Status



DEPTH FIRST SEARCH

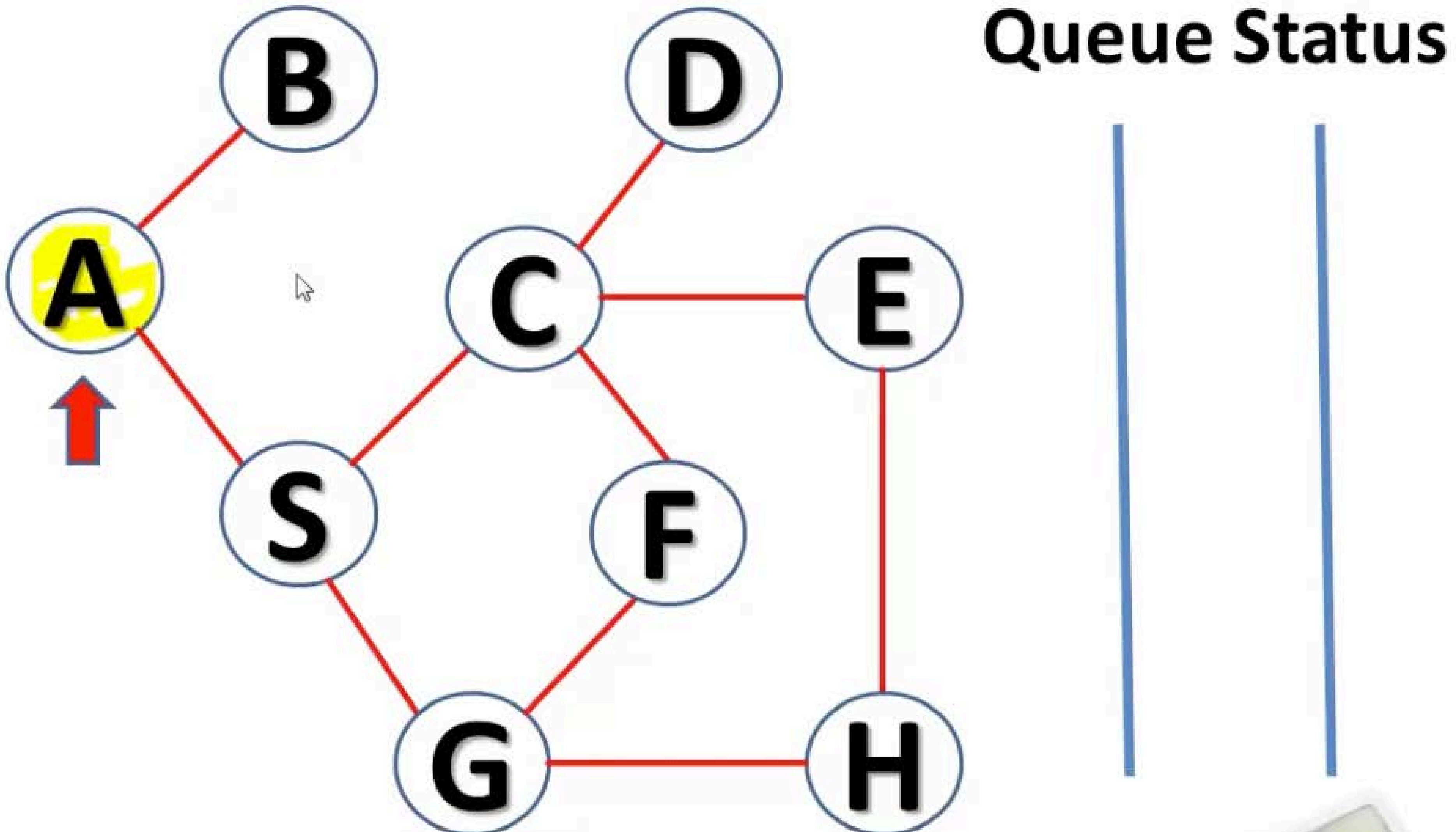


Stack Status

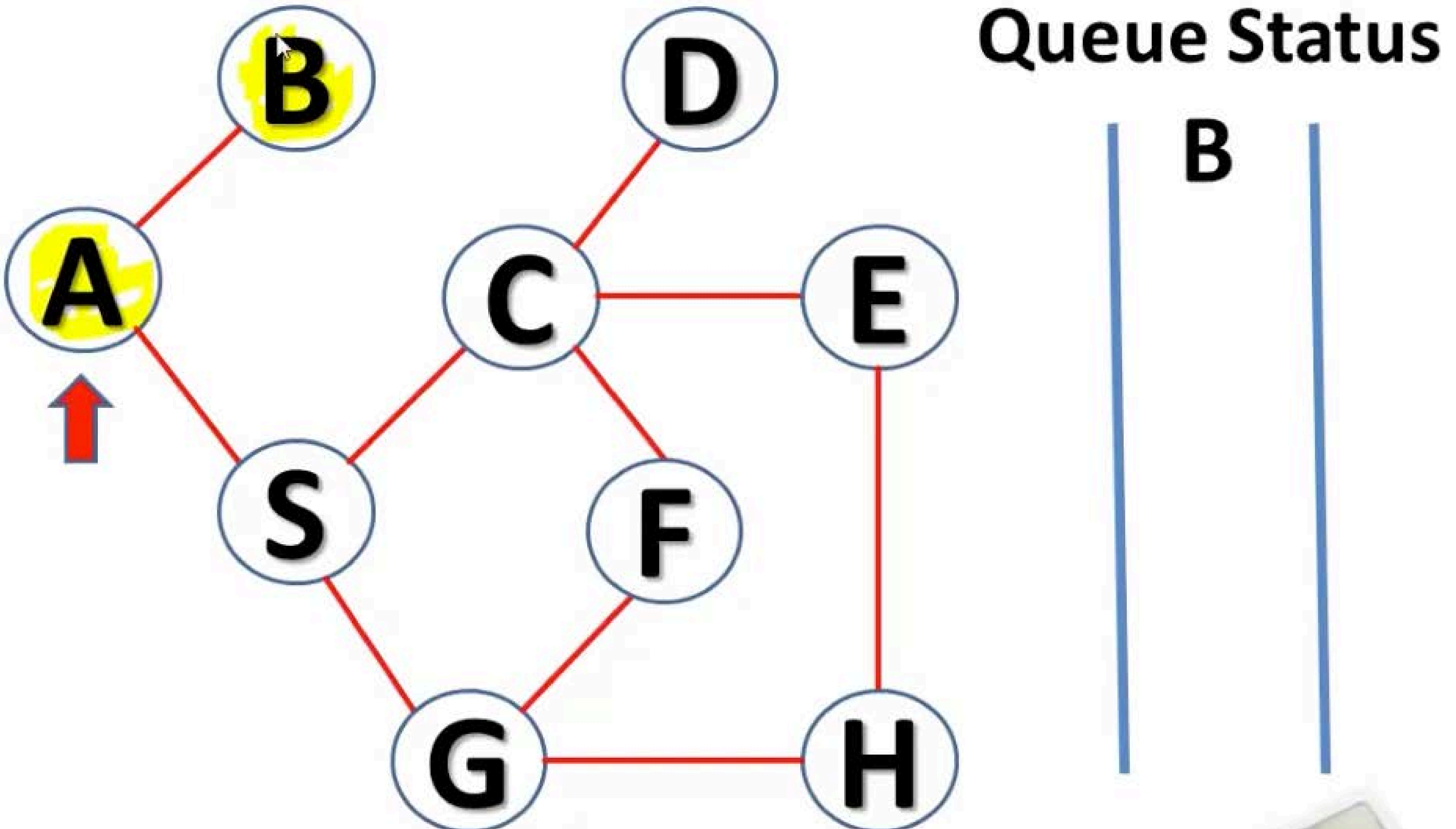


2. BFS WITH QUEUE

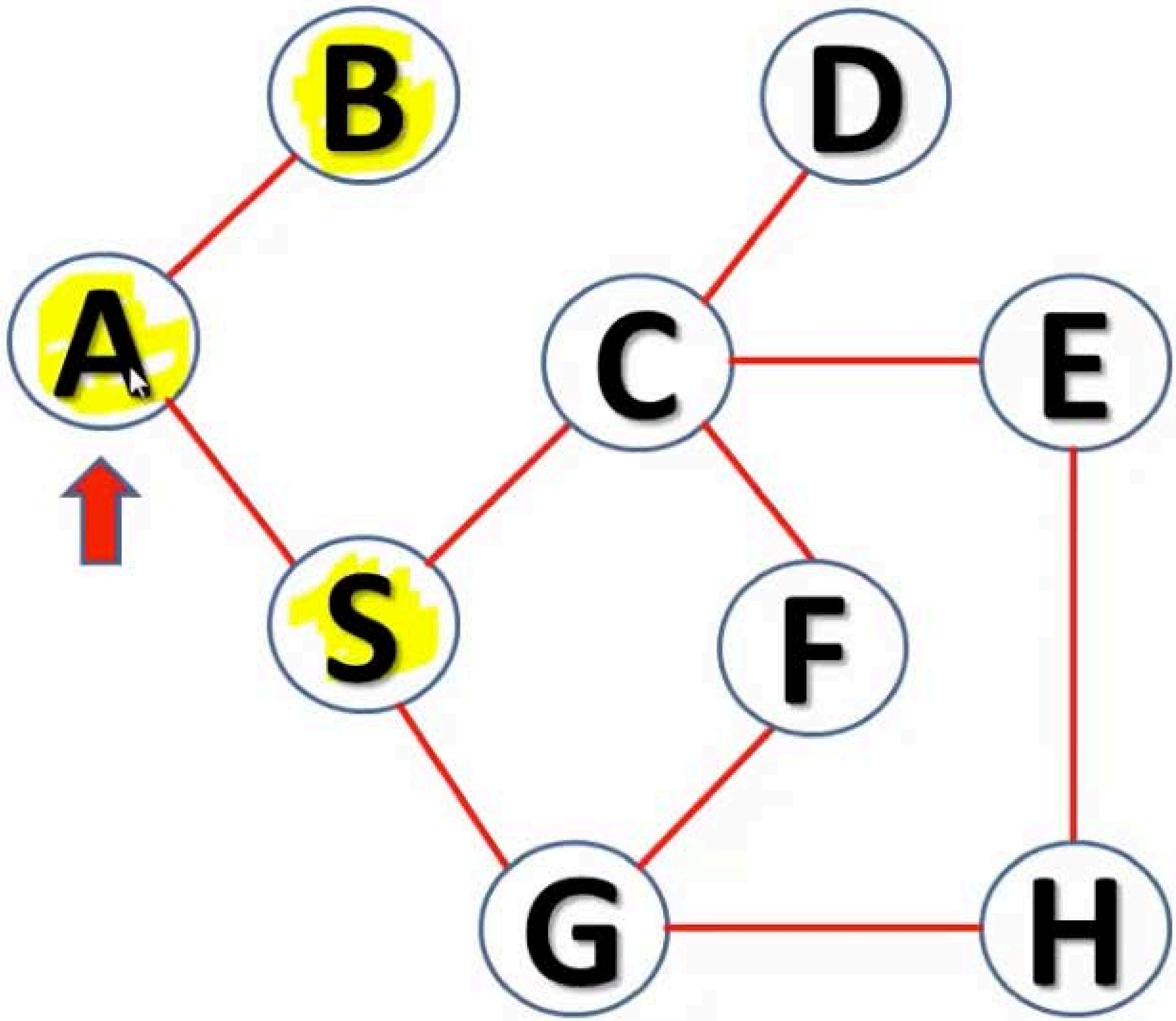
BREADTH FIRST SEARCH



BREADTH FIRST SEARCH



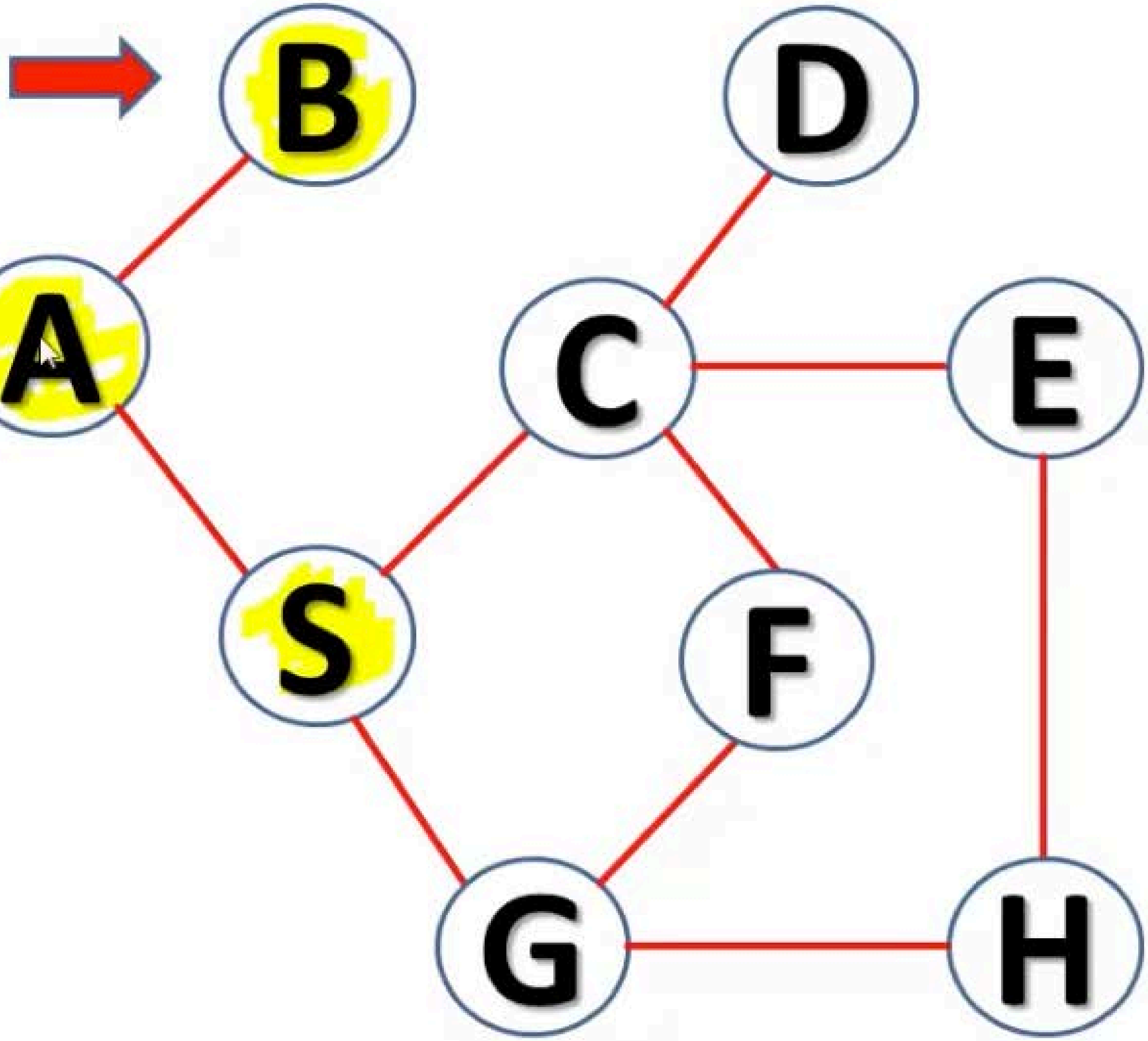
BREADTH FIRST SEARCH

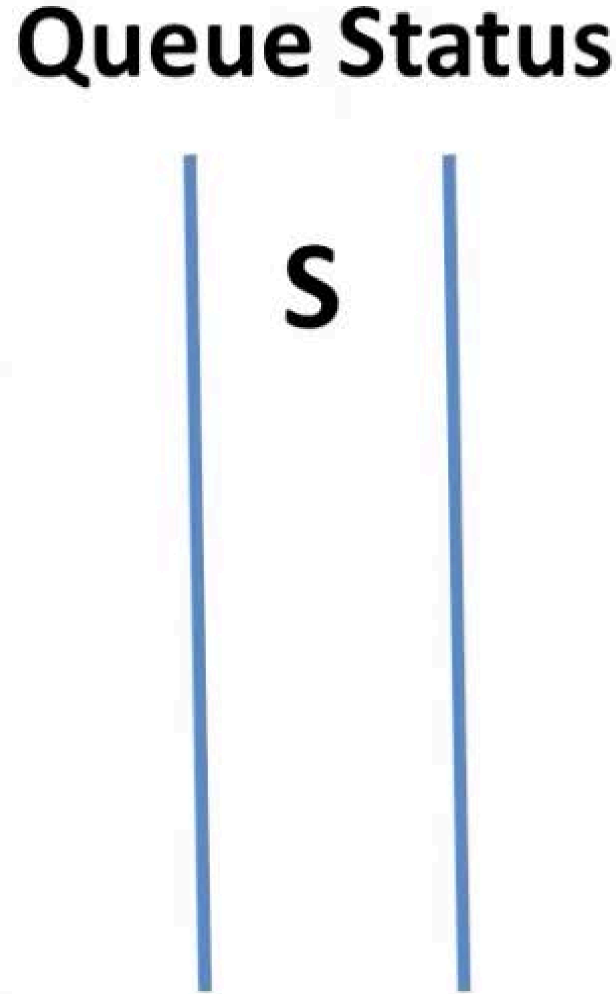


Queue Status

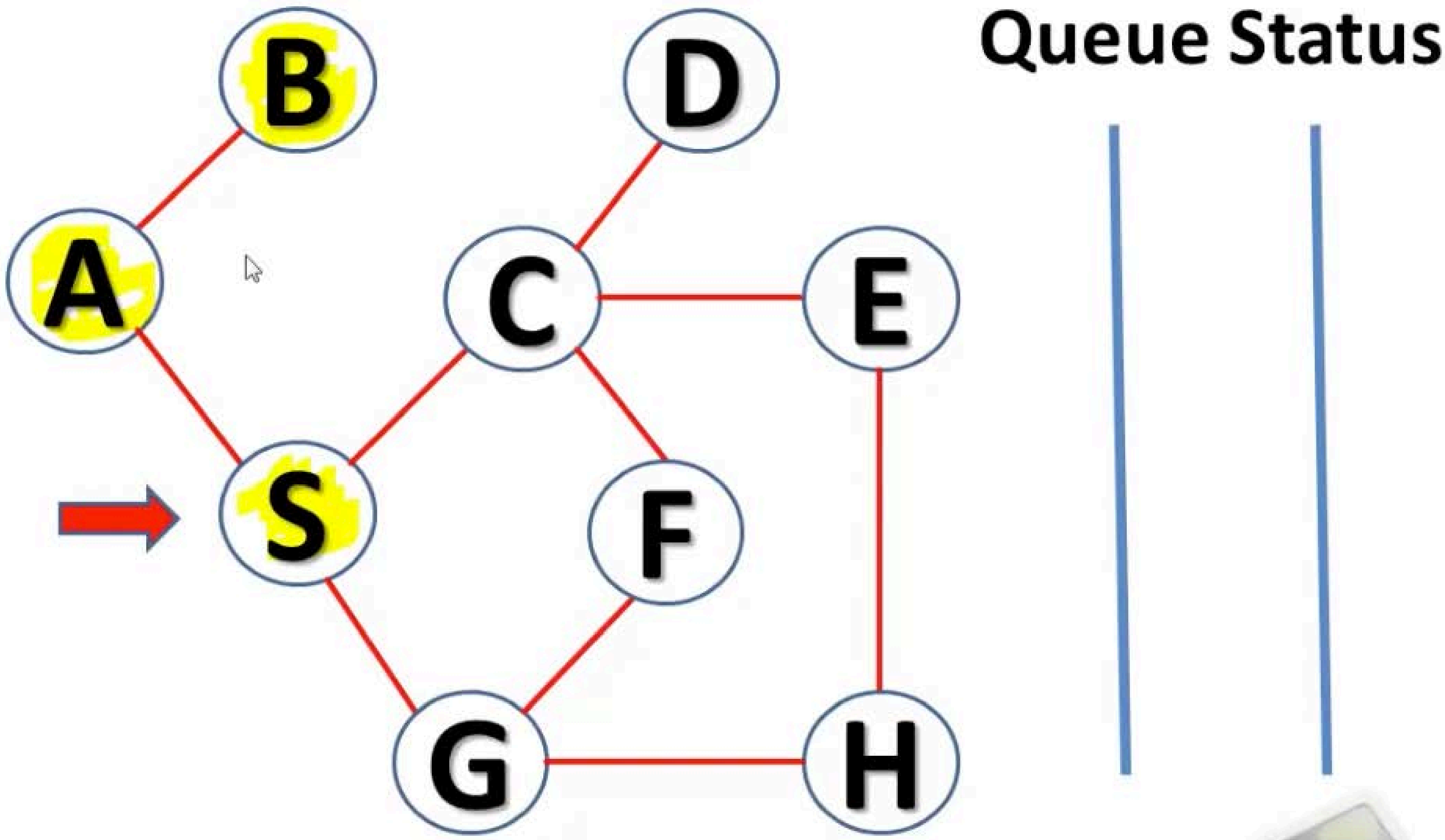
B
S

BREADTH FIRST SEARCH

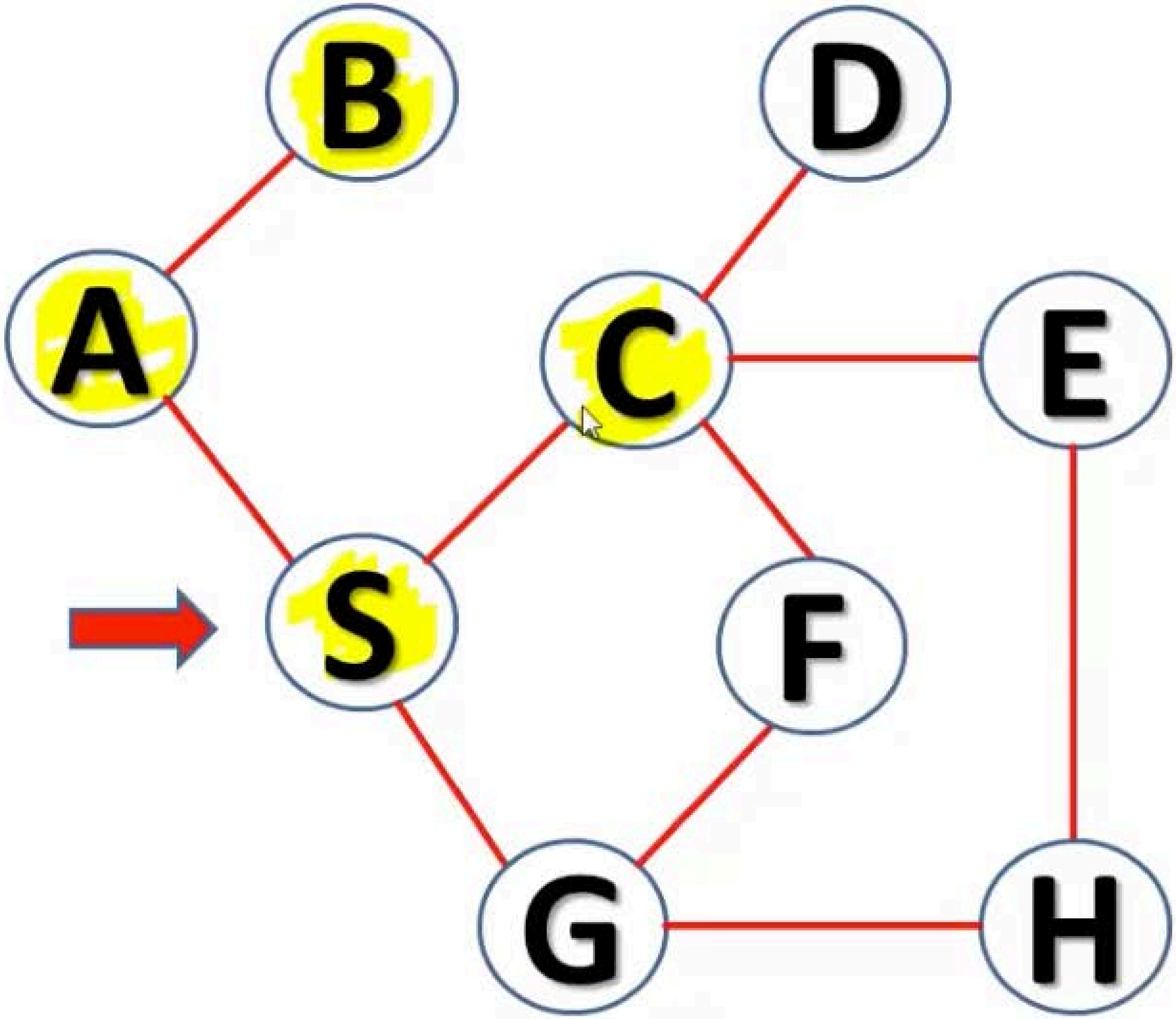




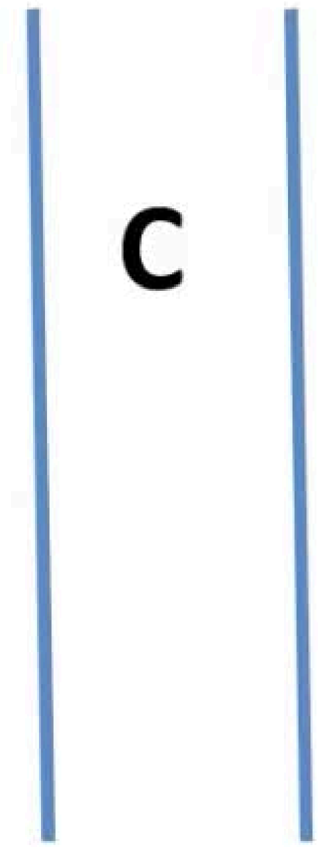
BREADTH FIRST SEARCH



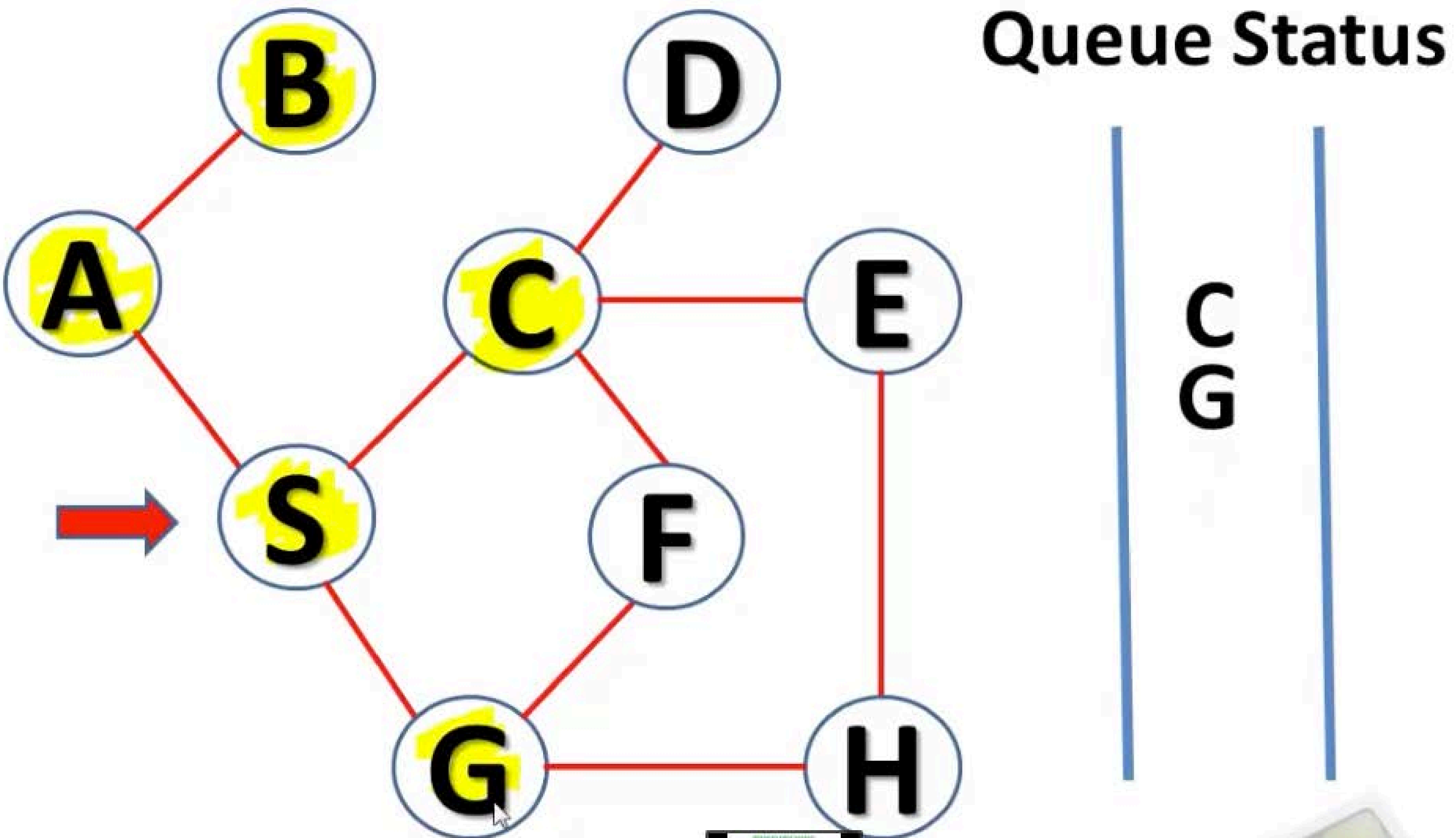
BREADTH FIRST SEARCH



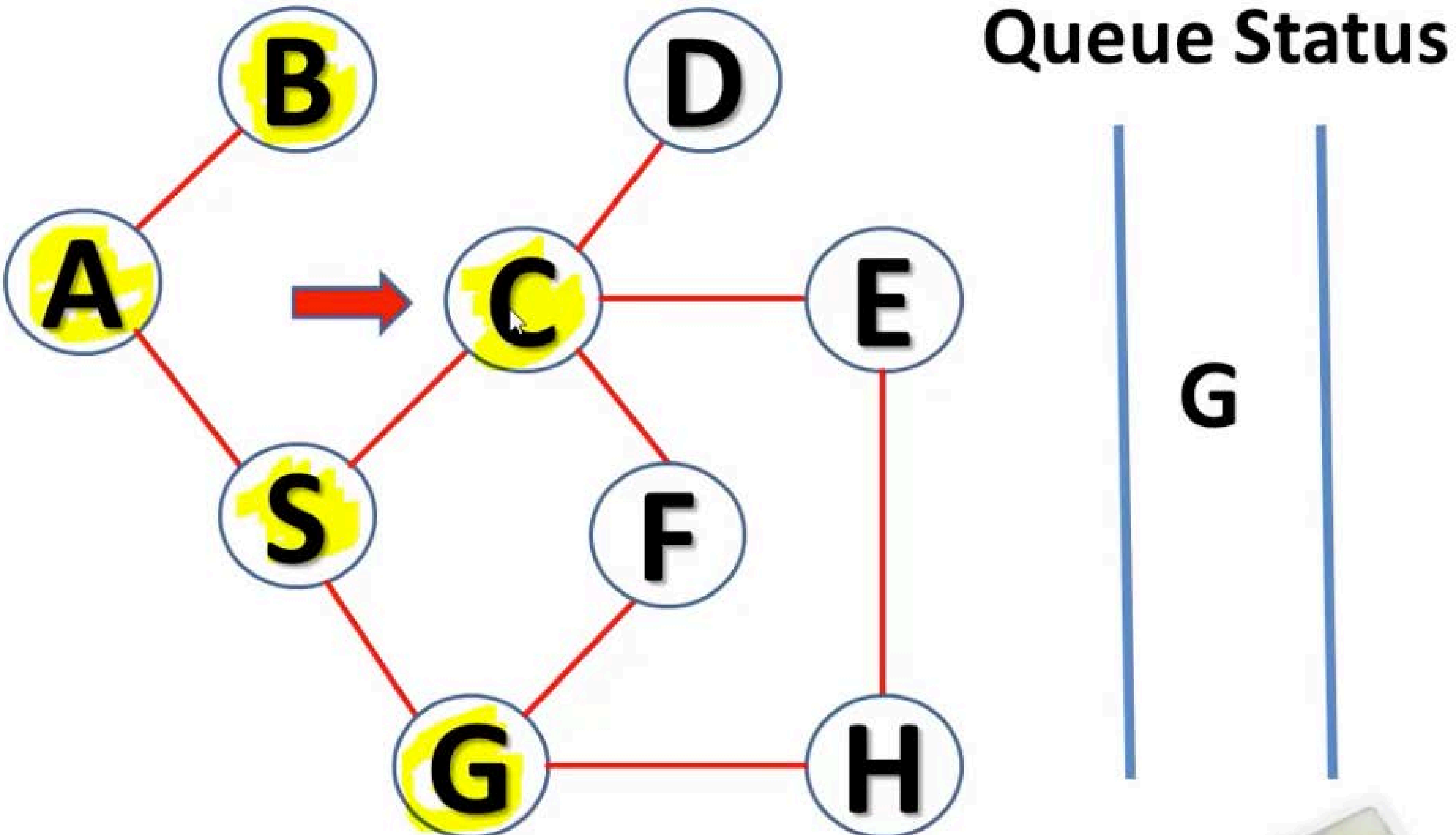
Queue Status



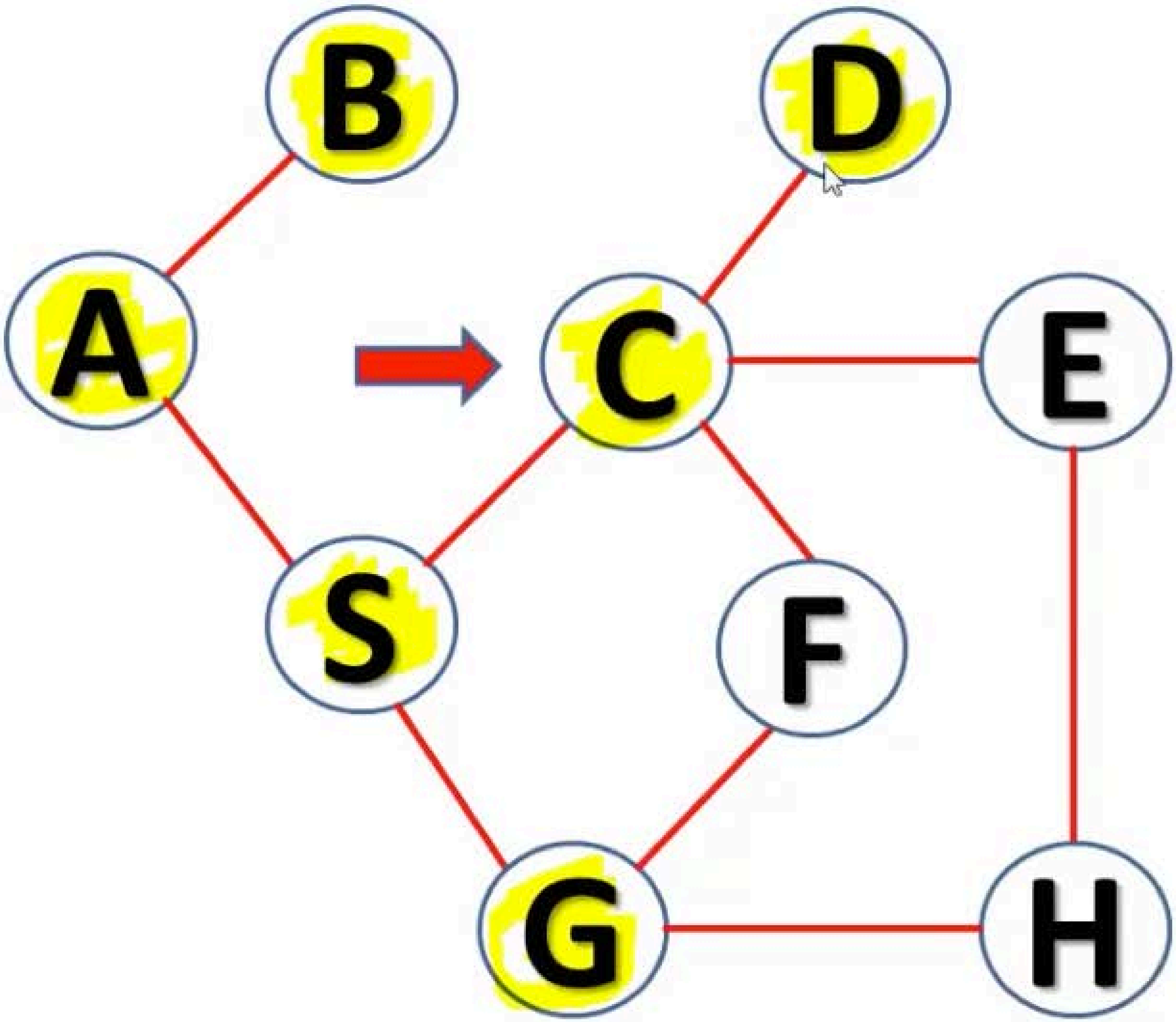
BREADTH FIRST SEARCH



BREADTH FIRST SEARCH



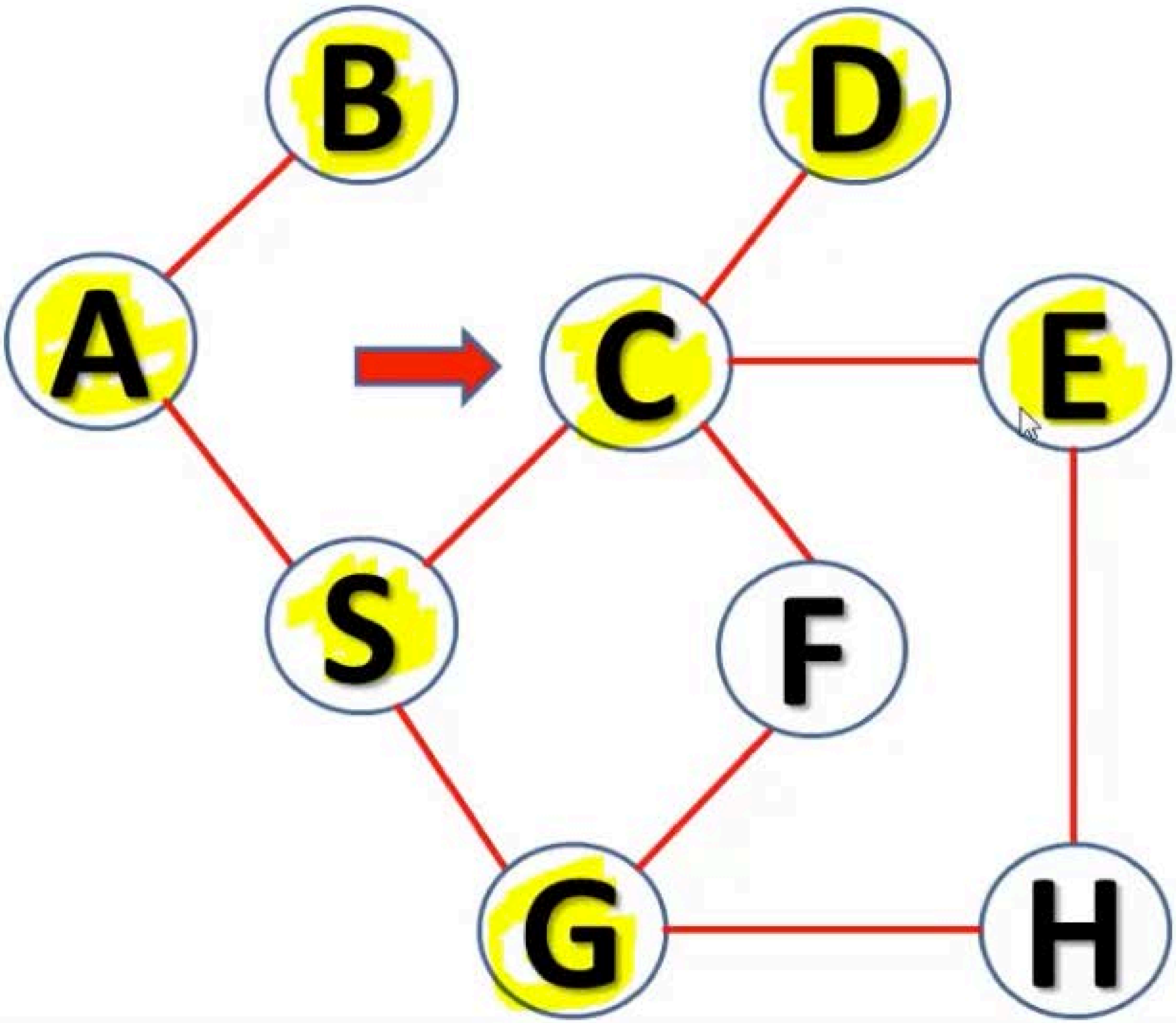
BREADTH FIRST SEARCH



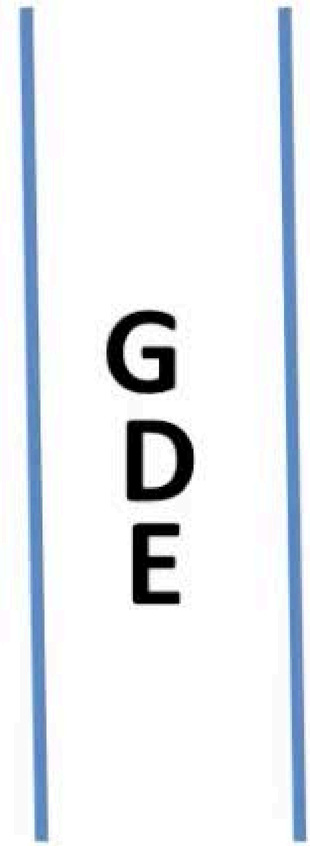
Queue Status

G
D

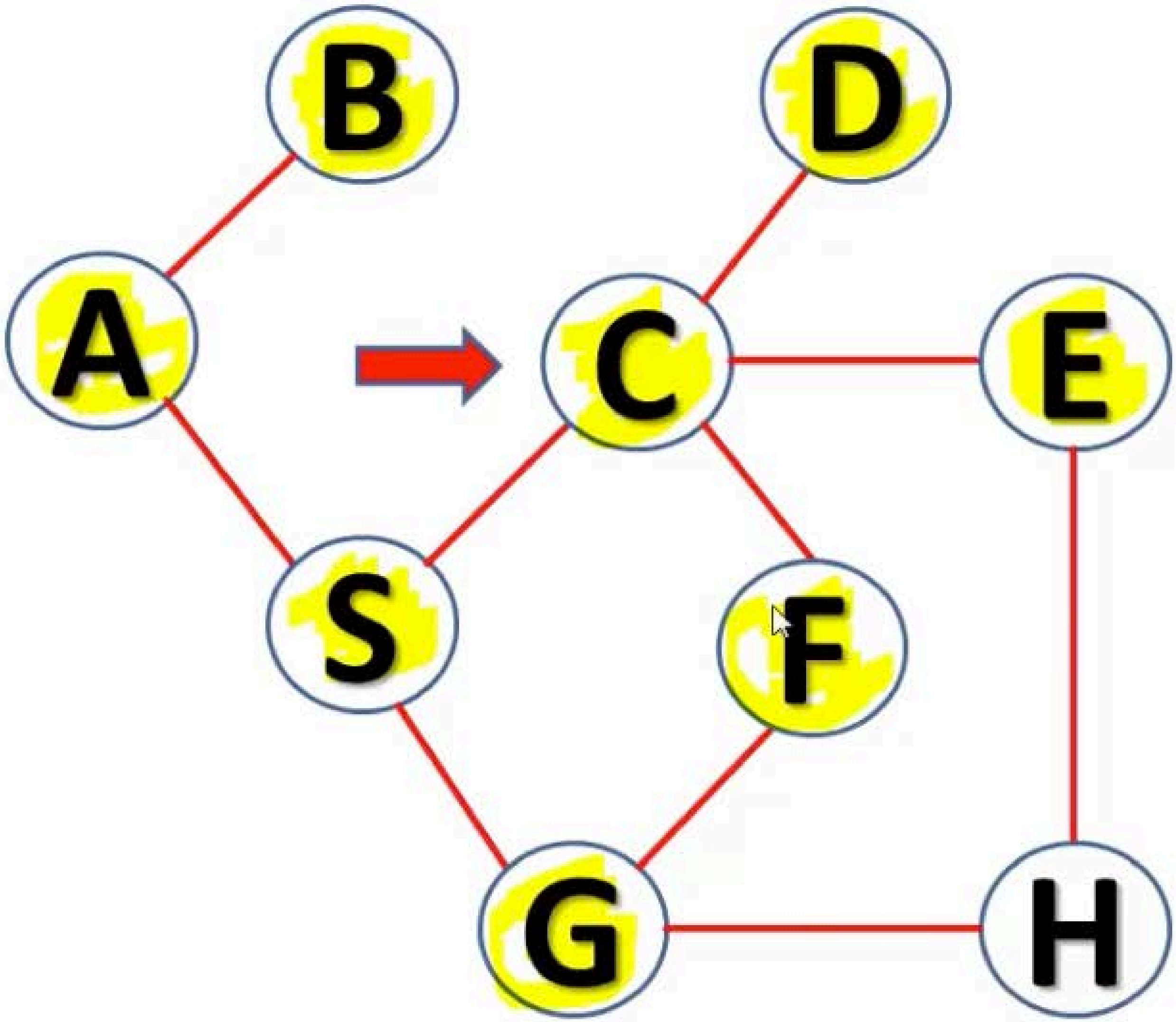
BREADTH FIRST SEARCH



Queue Status



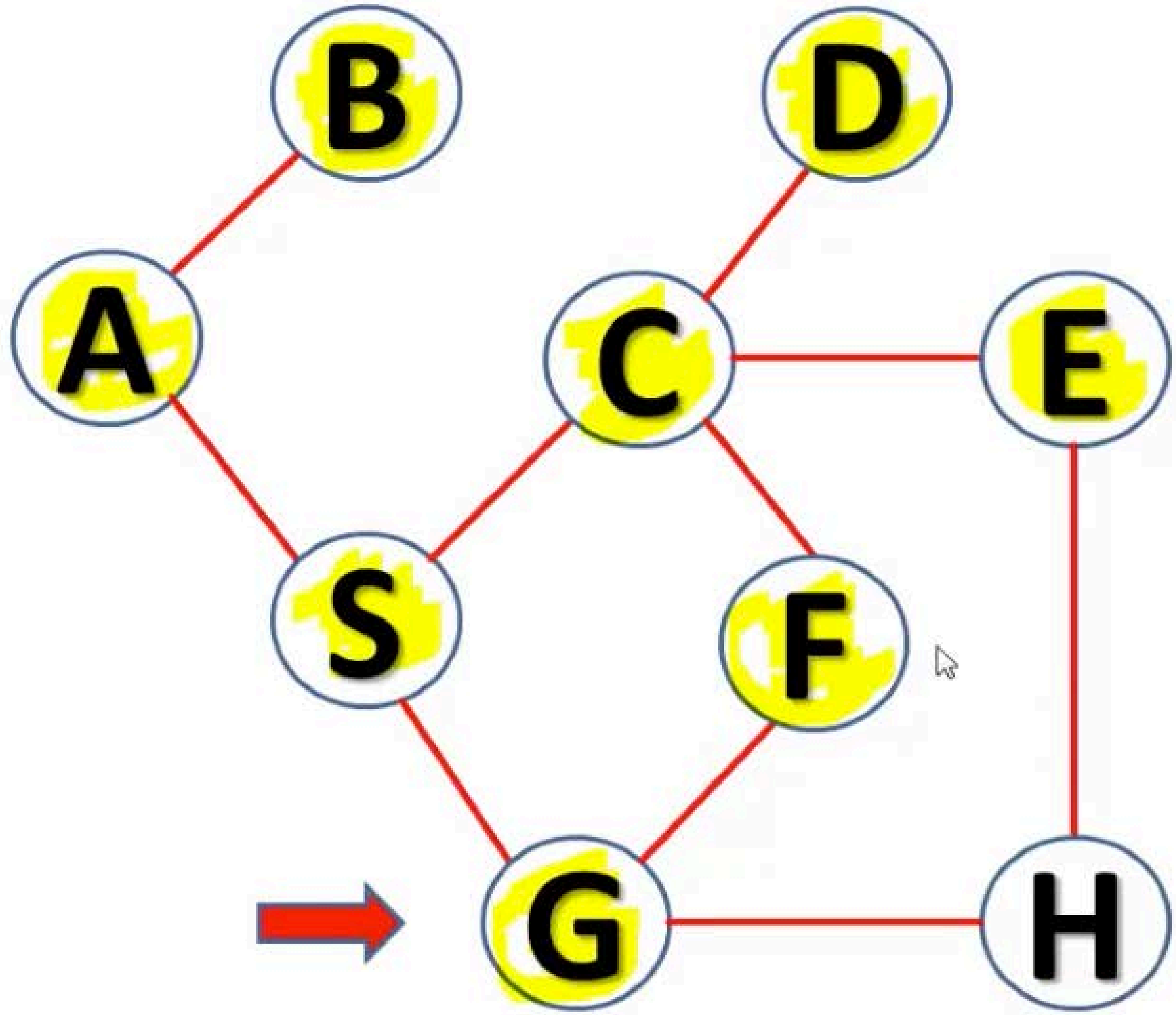
BREADTH FIRST SEARCH



Queue Status

G
D
E
F

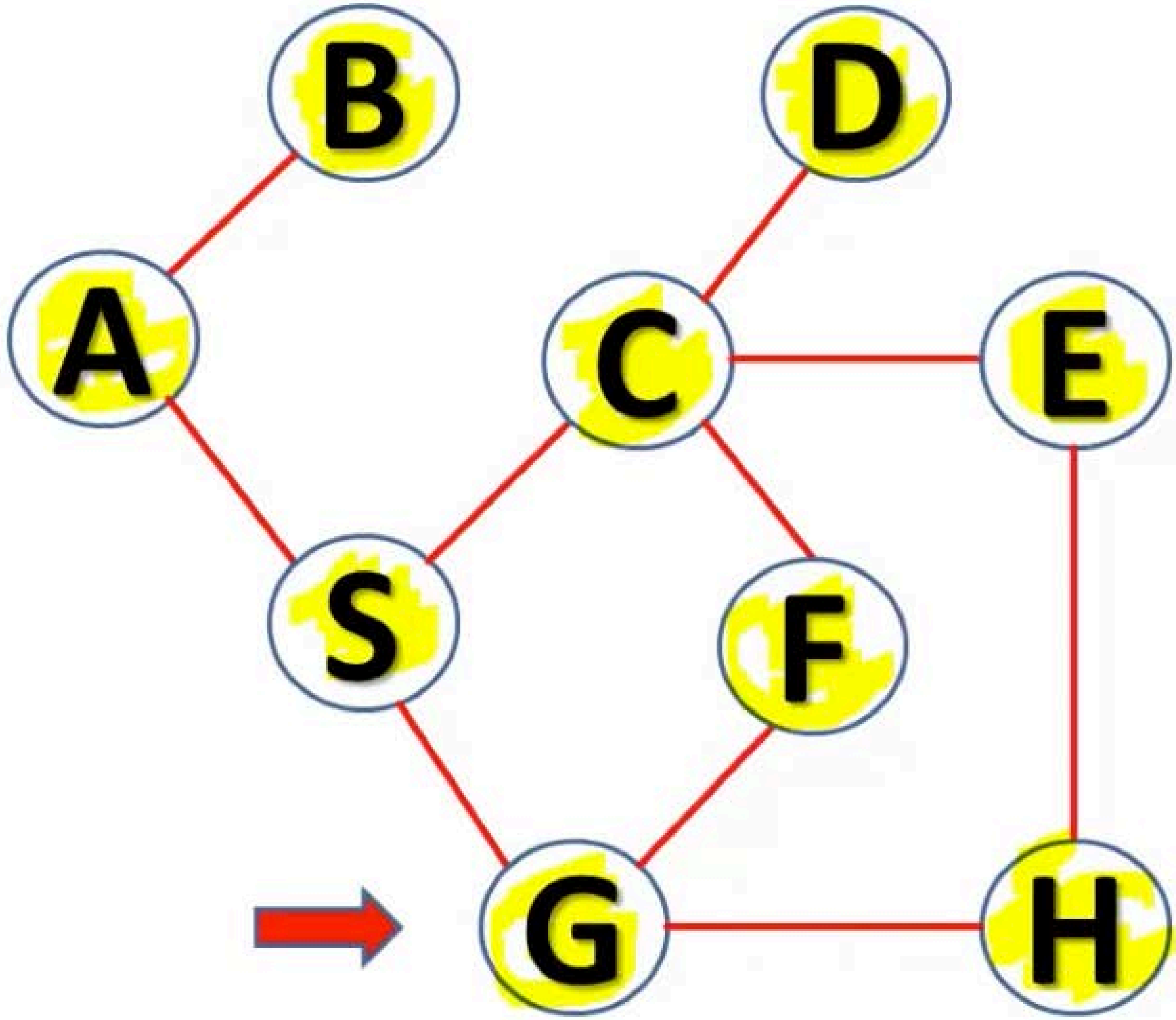
BREADTH FIRST SEARCH



Queue Status

D
E
F

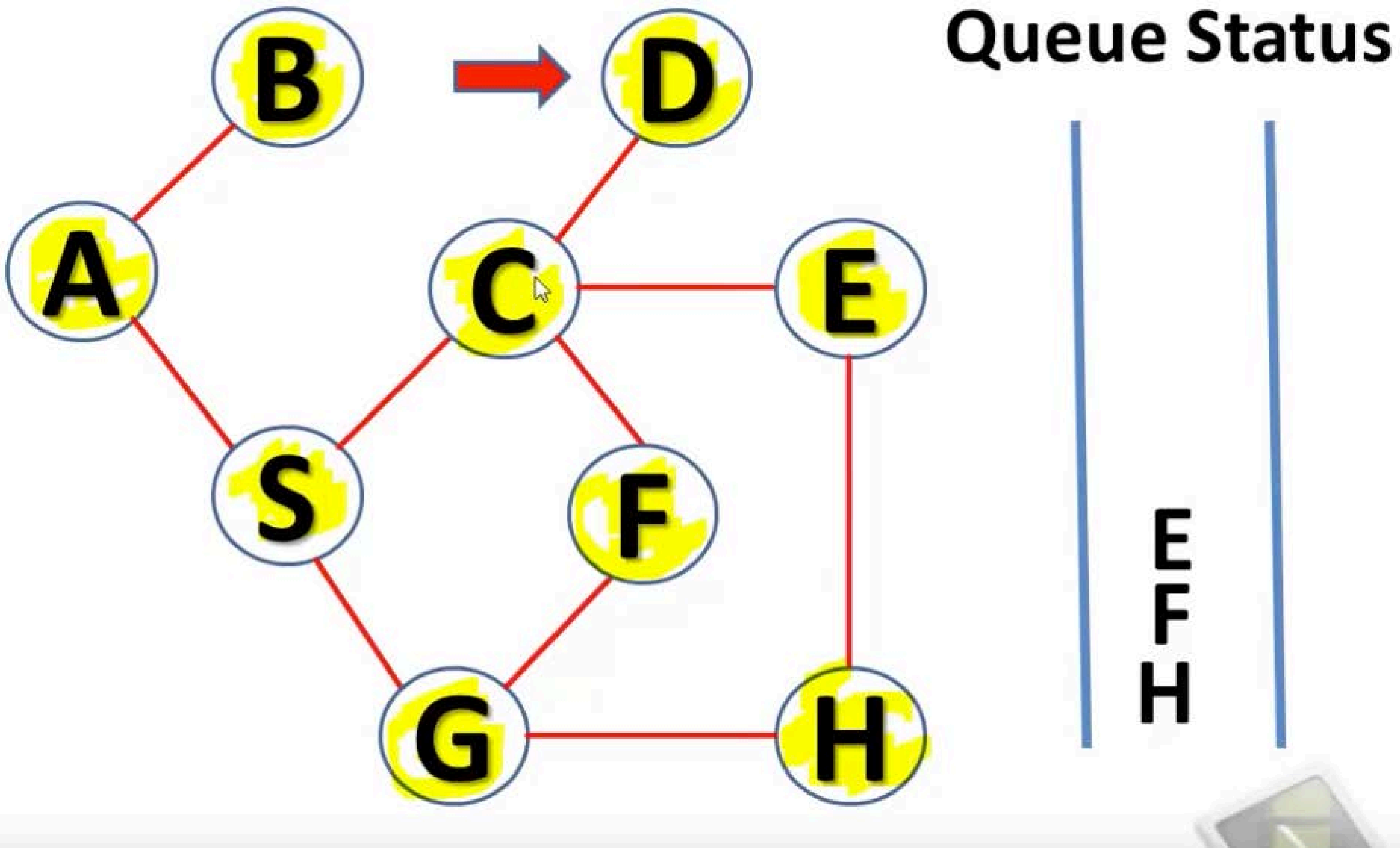
BREADTH FIRST SEARCH



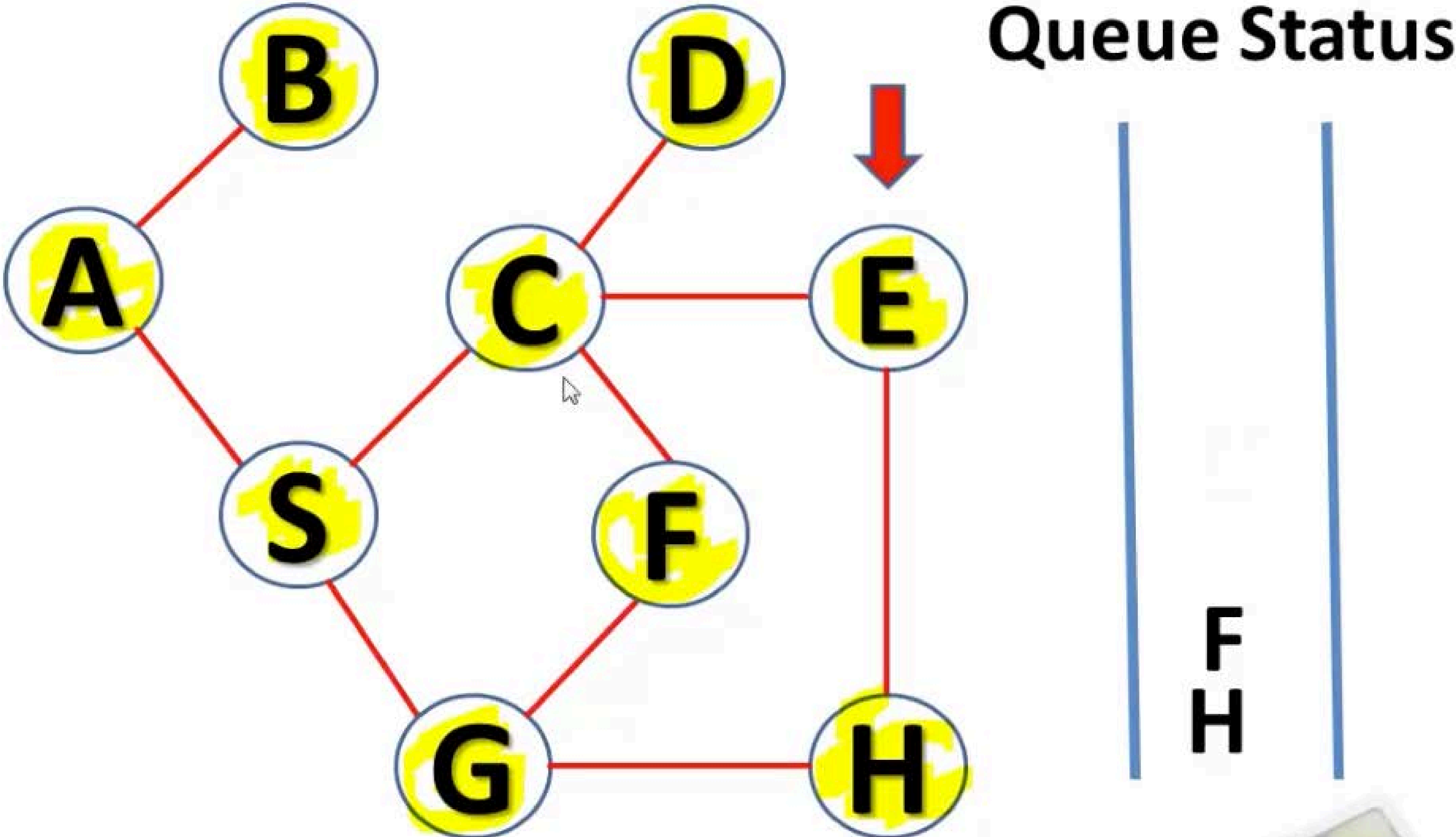
Queue Status

D
E
F
H

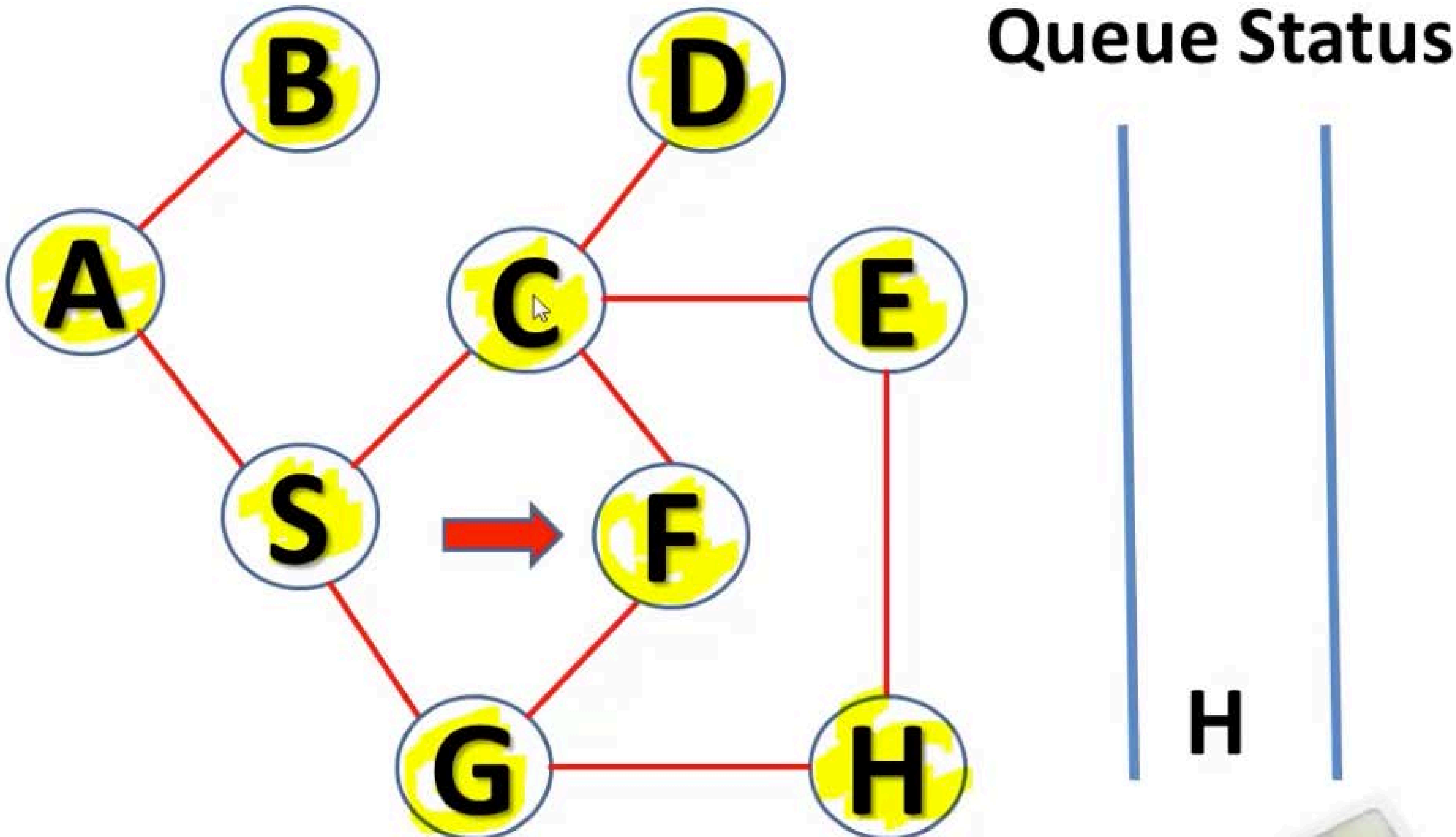
BREADTH FIRST SEARCH



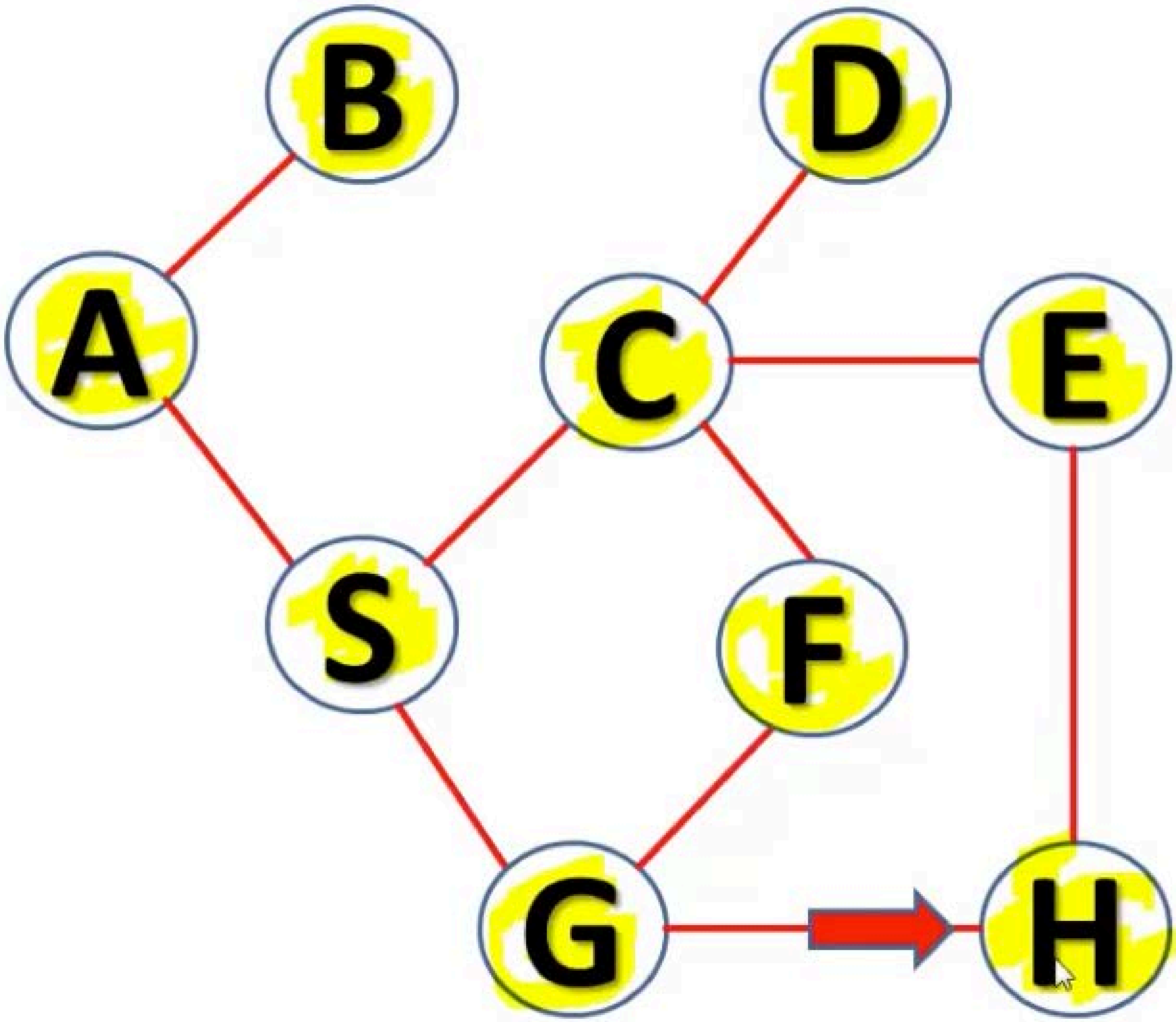
BREADTH FIRST SEARCH



BREADTH FIRST SEARCH



BREADTH FIRST SEARCH



Queue Status