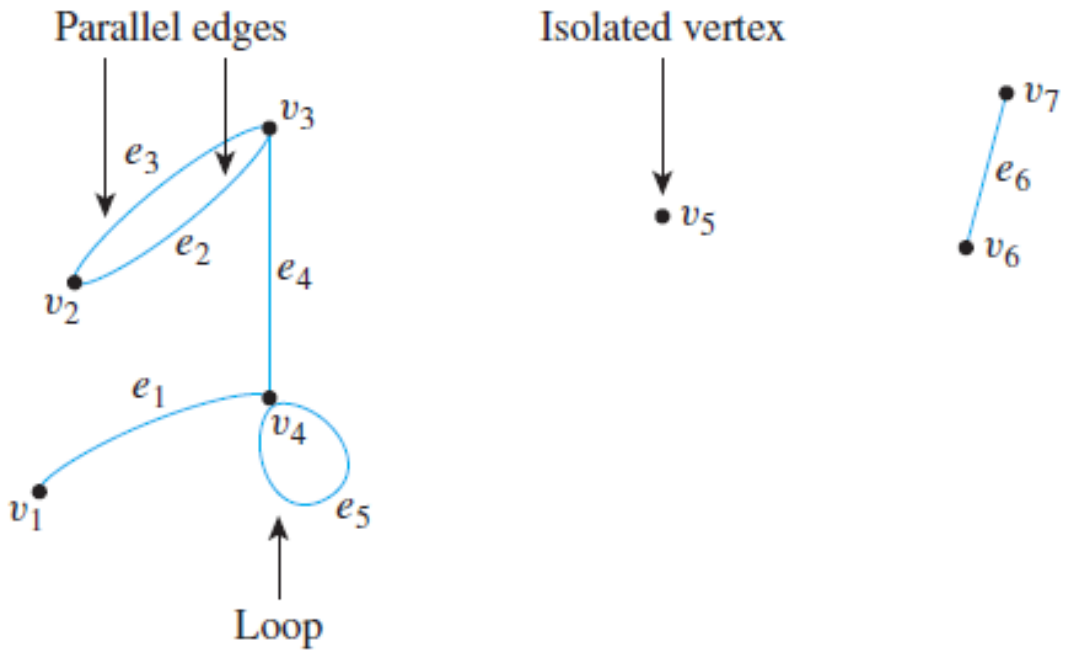
10 Graphs

The core nonlinear abstract data structures of computer science

Basic Terminologies



Graphs

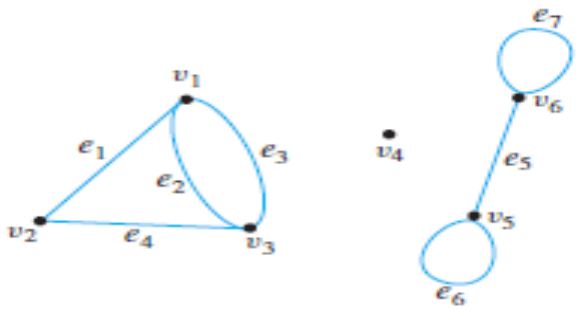
A **graph** G consists of two finite sets: a nonempty set $V(G)$ of **vertices** and a set $E(G)$ of **edges,** where each edge is associated with a set consisting of either one or two vertices called its **endpoints.** The correspondence from edges to endpoints is called the **edge-endpoint function.**

An edge with just one endpoint is called a **loop,** and two or more distinct edges with the same set of endpoints are said to be **parallel.** An edge is said to **connect** its endpoints; two vertices that are connected by an edge are called **adjacent;** and a vertex that is an endpoint of a loop is said to be **adjacent to itself.**

An edge is said to be **incident on** each of its endpoints, and two edges incident on the same endpoint are called **adjacent.** A vertex on which no edges are incident is called **isolated.**

Example

Consider the following graph:



a. Write the vertex set and the edge set, and give a table showing the edge-endpoint function.

b. Find all edges that are incident on v_1, all vertices that are adjacent to v_1, all edges that are adjacent to e_1, all loops, all parallel edges, all vertices that are adjacent to themselves, and all isolated vertices.

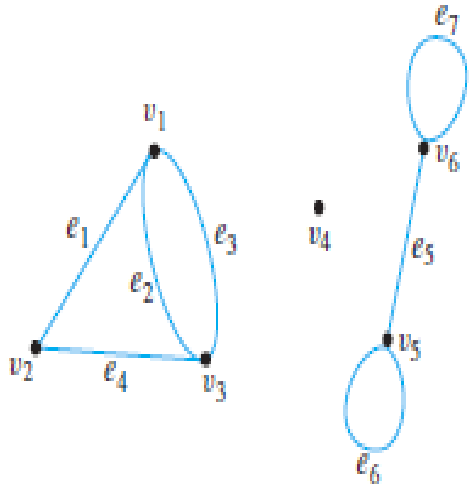
Solution

a. vertex set $= \{v_1, v_2, v_3, v_4, v_5, v_6\}$
edge set $= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
edge-endpoint function:

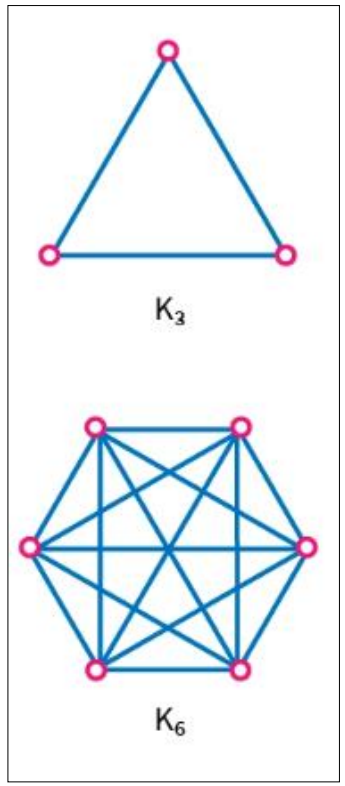
Edge	Endpoints
e_1	$\{v_1, v_2\}$
e_2	$\{v_1, v_3\}$
e_3	$\{v_1, v_3\}$
e_4	$\{v_2, v_3\}$
e_5	$\{v_5, v_6\}$
e_6	$\{v_5\}$
e_7	$\{v_6\}$

Note that the isolated vertex v_4 does not appear in this table. Although each edge must have either one or two endpoints, a vertex need not be an endpoint of an edge.

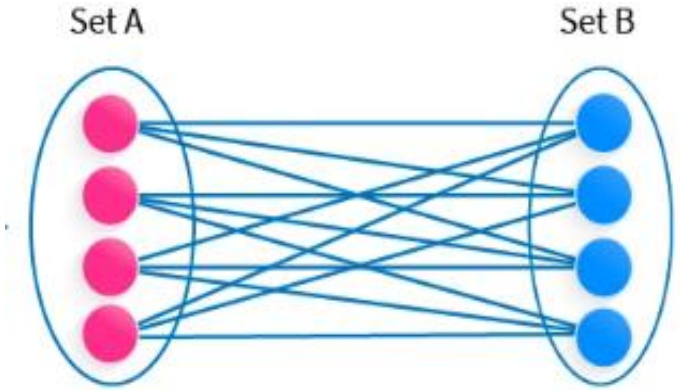
b. e_1, e_2, and e_3 are incident on v_1.
v_2 and v_3 are adjacent to v_1.
e_2, e_3, and e_4 are adjacent to e_1.
e_6 and e_7 are loops.
e_2 and e_3 are parallel.
v_5 and v_6 are adjacent to themselves.
v_4 is an isolated vertex.



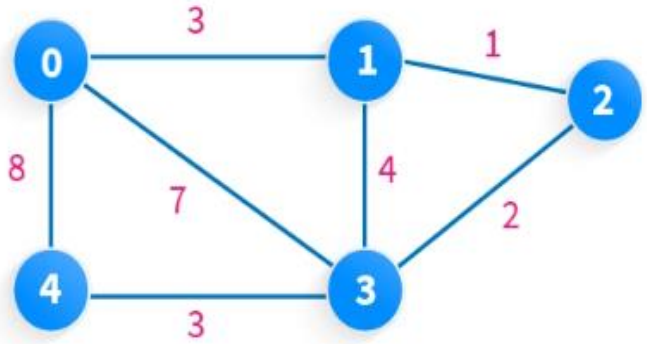
Types of Graphs



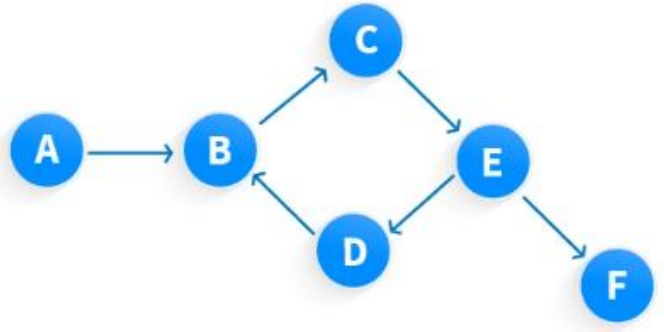
Complete Graph



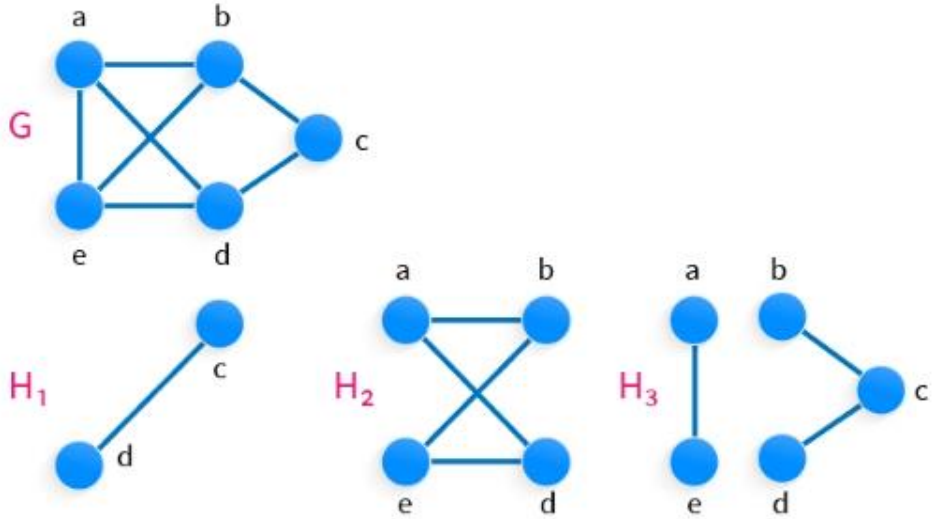
Bipartite Graph



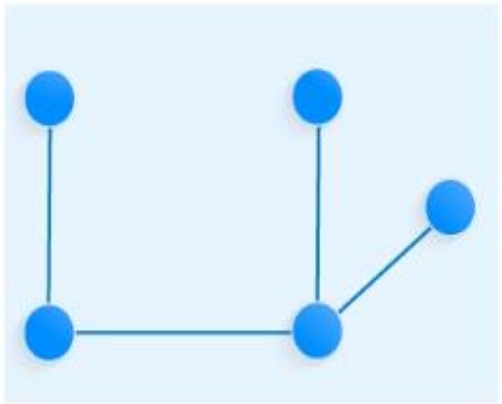
Weighted Graph



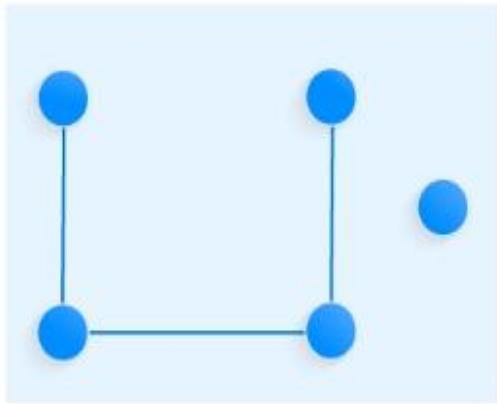
Directed Graph or Digraph



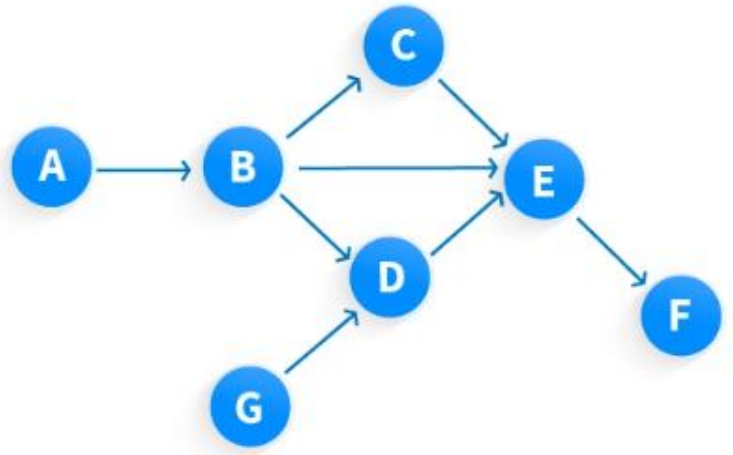
Subgraph



Connected Graph



Disconnected Graph



Directed Acyclic Graph (DAG)

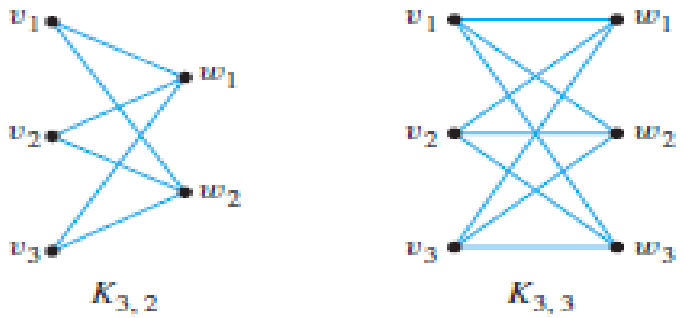
Bipartite Graph

Let m and n be positive integers. **A complete bipartite graph on (m, n) vertices,** denoted $\boldsymbol{K_{m,n}}$, is a simple graph with distinct vertices $v_1, v_2, \ldots, v_m$ and $w_1, w_2, \ldots, w_n$ that satisfies the following properties: For all $i, k = 1, 2, \ldots, m$ and for all $j, l = 1, 2, \ldots, n$,

1. There is an edge from each vertex v_i to each vertex w_j.
2. There is no edge from any vertex v_i to any other vertex v_k.
3. There is no edge from any vertex w_j to any other vertex w_l.

Complete Bipartite Graphs: $K_{3,2}$ and $K_{3,3}$

The complete bipartite graphs $K_{3,2}$ and $K_{3,3}$ are illustrated below.



Subgraph

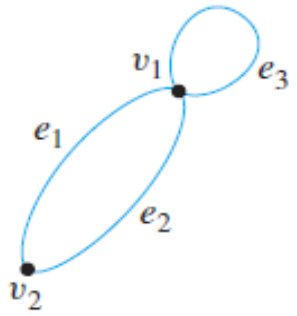
• Definition

A graph H is said to be a **subgraph** of a graph G if, and only if, every vertex in H is also a vertex in G, every edge in H is also an edge in G, and every edge in H has the same endpoints as it has in G.

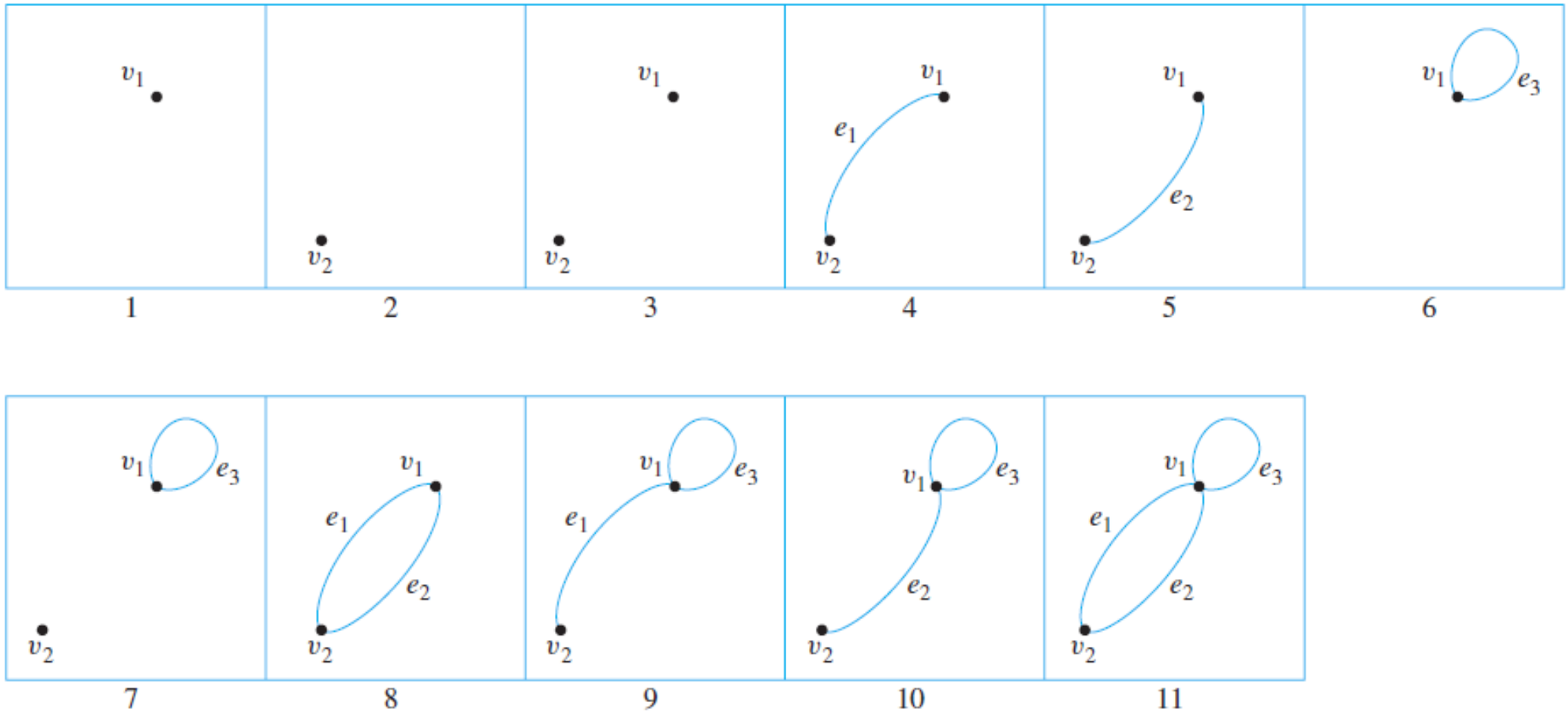
1 Subgraphs

List all subgraphs of the graph G with vertex set $\{v_1, v_2\}$ and edge set $\{e_1, e_2, e_3\}$, where the endpoints of e_1 are v_1 and v_2, the endpoints of e_2 are v_1 and v_2, and e_3 is a loop at v_1.

Solution G can be drawn as shown below.



There are 11 subgraphs of G, which can be grouped according to those that do not have any edges, those that have one edge, those that have two edges, and those that have three edges. The 11 subgraphs are shown in Figure 10.1.4.

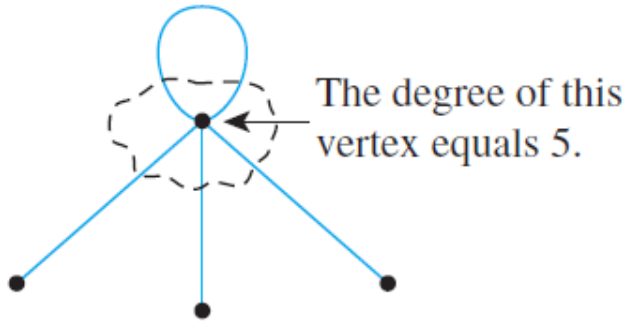


Degree of a Vertex

• Definition

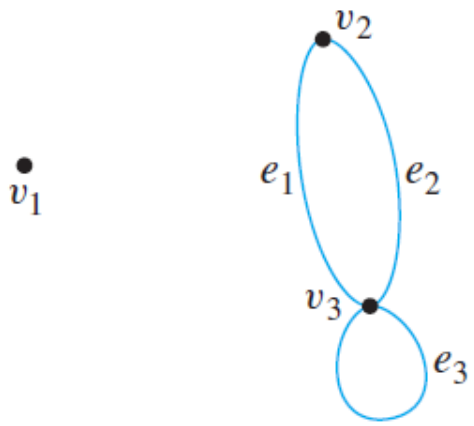
Let G be a graph and v a vertex of G. The **degree of v**, denoted **$\deg(v)$**, equals the number of edges that are incident on v, with an edge that is a loop counted twice. The **total degree of G** is the sum of the degrees of all the vertices of G.

- Number of edge segment of the vertex that stick out of the vertex.
- Loop is counted twice.
- the sum of the degrees of all the vertices in a graph is twice the number of edges of the graph.



Example

Find the degree of each vertex of the graph G shown below. Then find the total degree of G.



Solution

$\deg(v_1) = 0$ since no edge is incident on v_1 (v_1 is isolated).

$\deg(v_2) = 2$ since both e_1 and e_2 are incident on v_2.

$\deg(v_3) = 4$ since e_1 and e_2 are incident on v_3 and the loop e_3 is also incident on v_3 (and contributes 2 to the degree of v_3).

total degree of $G = \deg(v_1) + \deg(v_2) + \deg(v_3) = 0 + 2 + 4 = 6$. ■

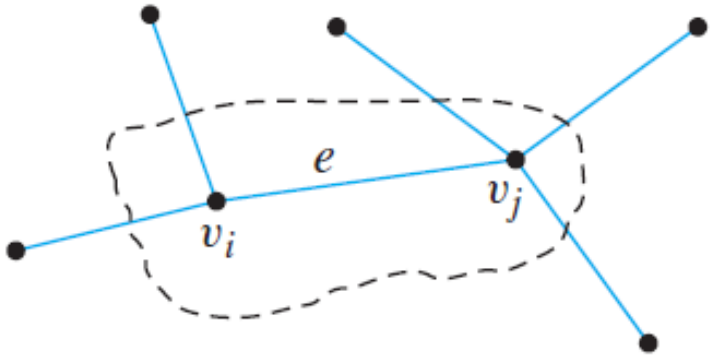
Theorem 10.1.1 The Handshake Theorem

If G is any graph, then the sum of the degrees of all the vertices of G equals twice the number of edges of G. Specifically, if the vertices of G are $v_1, v_2, \ldots, v_n$, where n is a nonnegative integer, then

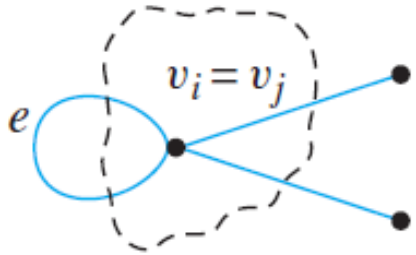
$$\begin{aligned}\text{the total degree of } G &= \deg(v_1) + \deg(v_2) + \cdots + \deg(v_n) \\ &= 2 \cdot (\text{the number of edges of } G).\end{aligned}$$

Proof:

Let G be a particular but arbitrarily chosen graph, and suppose that G has n vertices $v_1, v_2, \ldots, v_n$ and m edges, where n is a positive integer and m is a nonnegative integer. We claim that each edge of G contributes 2 to the total degree of G. For suppose e is an arbitrarily chosen edge with endpoints v_i and v_j. This edge contributes 1 to the degree of v_i and 1 to the degree v_j. As shown below, this is true even if $i = j$, because an edge that is a loop is counted twice in computing the degree of the vertex on which it is incident.



$i \neq j$



$i = j$

Therefore, e contributes 2 to the total degree of G. Since e was arbitrarily chosen, this shows that *each* edge of G contributes 2 to the total degree of G. Thus

$$\text{the total degree of } G = 2 \cdot (\text{the number of edges of } G).$$

Trails, Paths and Circuits

Let G be a graph, and let v and w be vertices in G.

A **walk from v to w** is a finite alternating sequence of adjacent vertices and edges of G. Thus a walk has the form

$$v_0 e_1 v_1 e_2 \cdots v_{n-1} e_n v_n,$$

where the v's represent vertices, the e's represent edges, $v_0 = v$, $v_n = w$, and for all $i = 1, 2, \ldots n$, v_{i-1} and v_i are the endpoints of e_i. The **trivial walk from v to v** consists of the single vertex v.

A **trail from v to w** is a walk from v to w that does not contain a repeated edge.

A **path from v to w** is a trail that does not contain a repeated vertex.

A **closed walk** is a walk that starts and ends at the same vertex.

A **circuit** is a closed walk that contains at least one edge and does not contain a repeated edge.

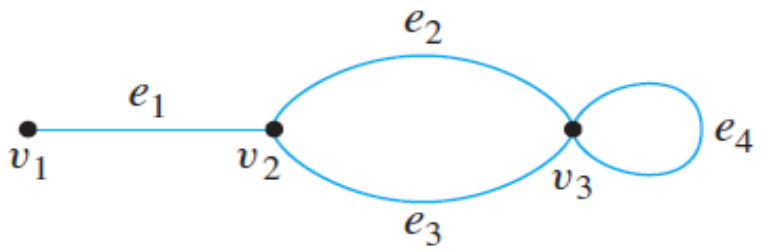
A **simple circuit** is a circuit that does not have any other repeated vertex except the first and last.

Definition Summary

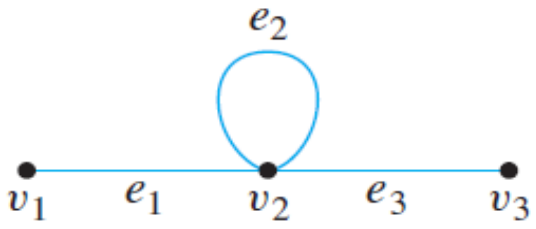
	Repeated Edge?	Repeated Vertex?	Starts and Ends at Same Point?	Must Contain at Least One Edge?
Walk	allowed	allowed	allowed	no
Trail	no	allowed	allowed	no
Path	no	no	no	no
Closed walk	allowed	allowed	yes	no
Circuit	no	allowed	yes	yes
Simple circuit	no	first and last only	yes	yes

Notation for Walks

a. In the graph below, the notation $e_1e_2e_4e_3$ refers unambiguously to the following walk: $v_1e_1v_2e_2v_3e_4v_3e_3v_2$. On the other hand, the notation e_1 is ambiguous if used to refer to a walk. It could mean either $v_1e_1v_2$ or $v_2e_1v_1$.



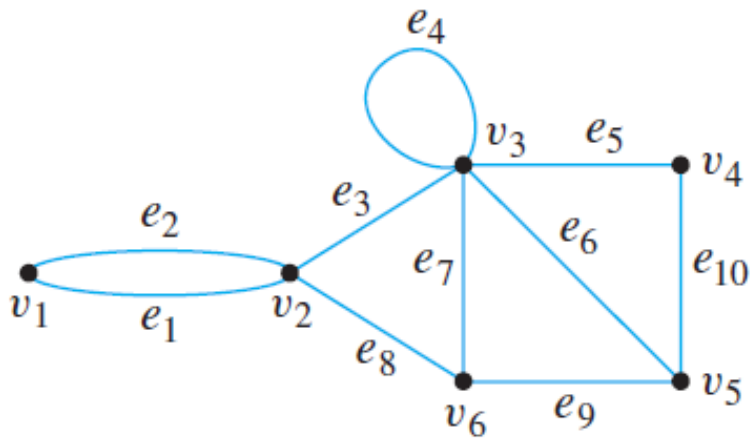
b. In the graph of part (a), the notation v_2v_3 is ambiguous if used to refer to a walk. It could mean $v_2e_2v_3$ or $v_2e_3v_3$. On the other hand, in the graph below, the notation $v_1v_2v_2v_3$ refers unambiguously to the walk $v_1e_1v_2e_2v_2e_3v_3$.



Example

In the graph below, determine which of the following walks are trails, paths, circuits, or simple circuits.

a. $v_1e_1v_2e_3v_3e_4v_3e_5v_4$
b. $e_1e_3e_5e_5e_6$
c. $v_2v_3v_4v_5v_3v_6v_2$
d. $v_2v_3v_4v_5v_6v_2$
e. $v_1e_1v_2e_1v_1$
f. v_1



Solution

a. This walk has a repeated vertex but does not have a repeated edge, so it is a trail from v_1 to v_4 but not a path.

b. This is just a walk from v_1 to v_5. It is not a trail because it has a repeated edge.

c. This walk starts and ends at v_2, contains at least one edge, and does not have a repeated edge, so it is a circuit. Since the vertex v_3 is repeated in the middle, it is not a simple circuit.

d. This walk starts and ends at v_2, contains at least one edge, does not have a repeated edge, and does not have a repeated vertex. Thus it is a simple circuit.

e. This is just a closed walk starting and ending at v_1. It is not a circuit because edge e_1 is repeated.

f. The first vertex of this walk is the same as its last vertex, but it does not contain an edge, and so it is not a circuit. It is a closed walk from v_1 to v_1. (It is also a trail from v_1 to v_1.)

Connectedness

- A graph is connected if it is possible to travel from any vertex to any other vertex along a sequence of adjacent edges of the graph.
- If take the negation of this definition, then that graph G is not connected if, and only if, there are two vertices of G that are not connected by any walk.

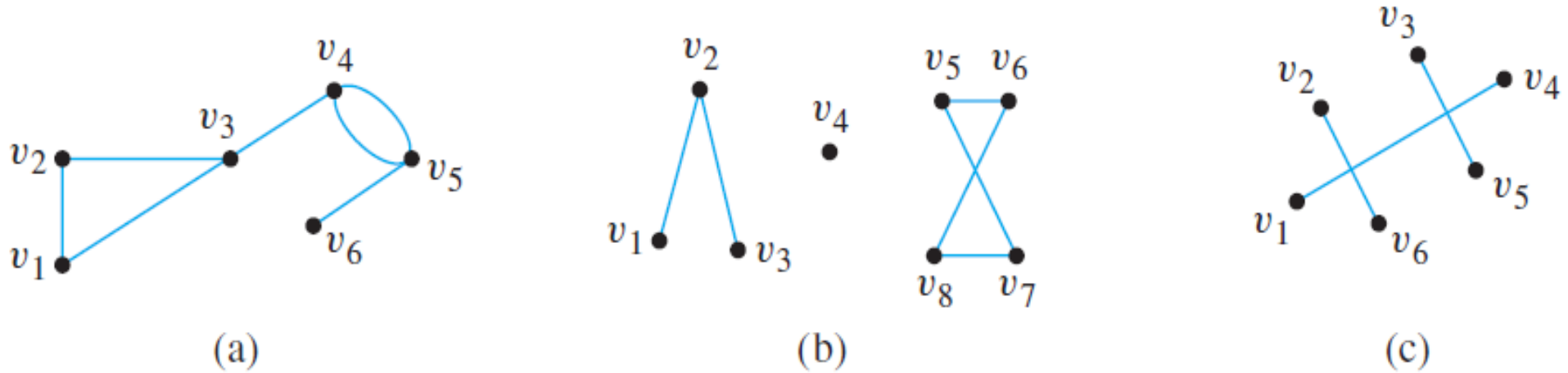
• Definition

Let G be a graph. Two **vertices v and w of G are connected** if, and only if, there is a walk from v to w. The **graph G is connected** if, and only if, given *any* two vertices v and w in G, there is a walk from v to w. Symbolically,

$$G \text{ is connected} \quad \Leftrightarrow \quad \forall \text{ vertices } v, w \in V(G), \exists \text{ a walk from } v \text{ to } w.$$

Example

Which of the following graphs are connected?



(a) Connected

(b) Disconnected

(c) Disconnected

Euler Circuits

• Definition

Let G be a graph. An **Euler circuit** for G is a circuit that contains every vertex and every edge of G. That is, an Euler circuit for G is a sequence of adjacent vertices and edges in G that has at least one edge, starts and ends at the same vertex, uses every vertex of G at least once, and uses every edge of G exactly once.

Example:

Theorem 10.2.3

If a graph G is connected and the degree of every vertex of G is a positive even integer, then G has an Euler circuit.

Proof:

Suppose that G is any connected graph and suppose that every vertex of G is a positive even integer. *[We must find an Euler circuit for G.]* Construct a circuit C by the following algorithm:

Step 1: Pick any vertex v of G at which to start.
[This step can be accomplished because the vertex set of G is nonempty by assumption.]

Step 2: Pick any sequence of adjacent vertices and edges, starting and ending at v and never repeating an edge. Call the resulting circuit C.
[This step can be performed for the following reasons: Since the degree of each vertex of G is a positive even integer, as each vertex of G is entered by traveling on one edge, either the vertex is v itself and there is no other unused edge

cent to v, or the vertex can be exited by traveling on another previously unused edge. Since the number of edges of the graph is finite (by definition of graph), the sequence of distinct edges cannot go on forever. The sequence can eventually return to v because the degree of v is a positive even integer, and so if an edge connects v to another vertex, there must be a different edge that connects back to v.]

Step 3: Check whether C contains every edge and vertex of G. If so, C is an Euler circuit, and we are finished. If not, perform the following steps.

Step 3a: Remove all edges of C from G and also any vertices that become isolated when the edges of C are removed. Call the resulting subgraph G'.

[Note that G' may not be connected (as illustrated in Figure 10.2.4), but every vertex of G' has positive, even degree (since removing the edges of C removes an even number of edges from each vertex, the difference of two even integers is even, and isolated vertices with degree 0 were removed.)]

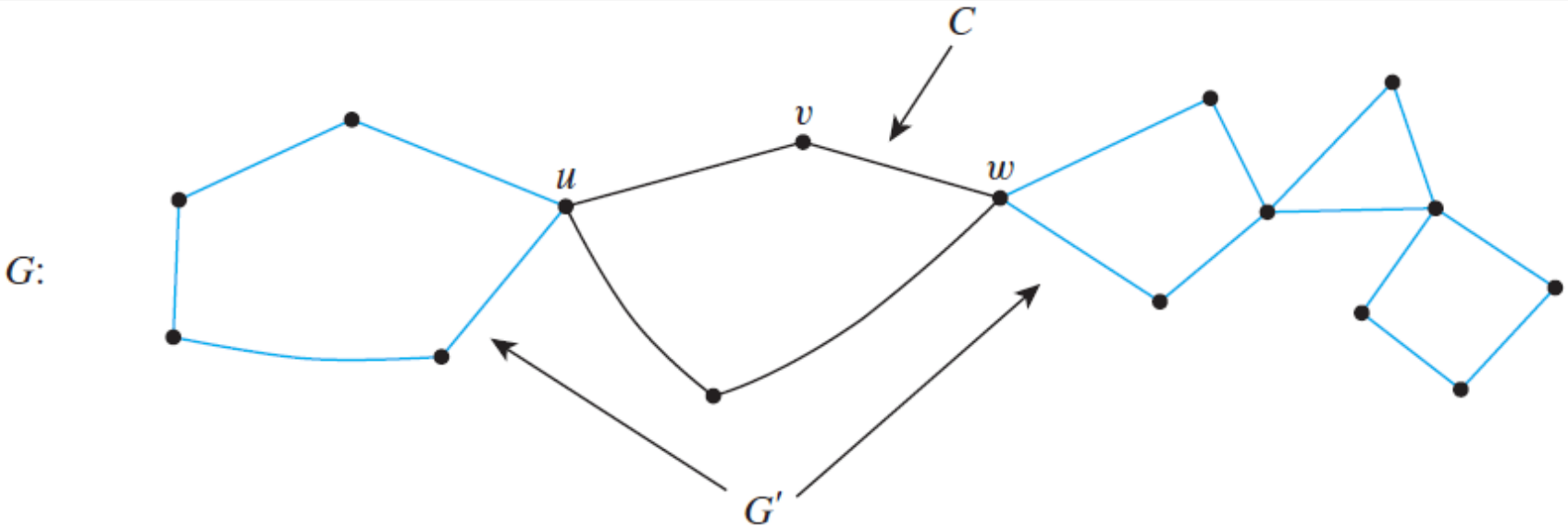


Figure 10.2.4

Step 3b: Pick any vertex w common to both C and G'.
[There must be at least one such vertex since G is connected. (See exercise 44.) (In Figure 10.2.4 there are two such vertices: u and w.)]

Step 3c: Pick any sequence of adjacent vertices and edges of G', starting and ending at w and never repeating an edge. Call the resulting circuit C'.
[This can be done since each vertex of G' has positive, even degree and G' is finite. See the justification for step 2.]

Step 3d: Patch C and C' together to create a new circuit C'' as follows: Start at v and follow C all the way to w. Then follow C' all the way back to w. After that, continue along the untraveled portion of C to return to v. *[The effect of executing steps 3c and 3d for the graph of Figure 10.2.4 is shown in Figure 10.2.5.]*

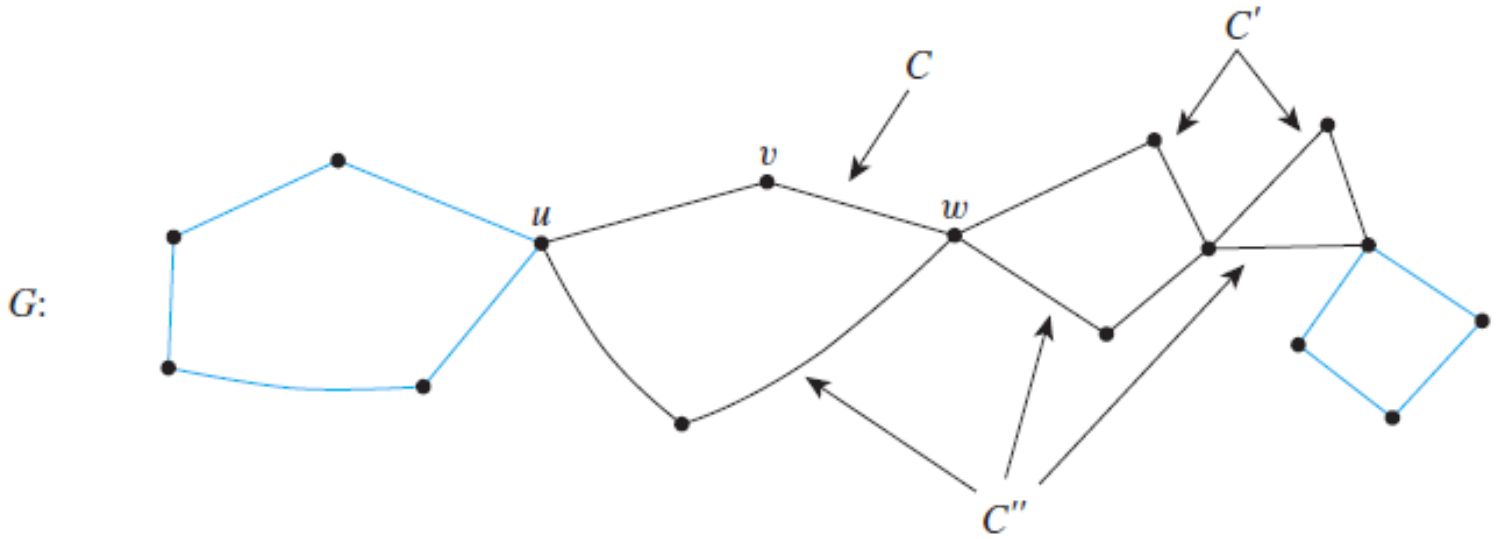


Figure 10.2.5

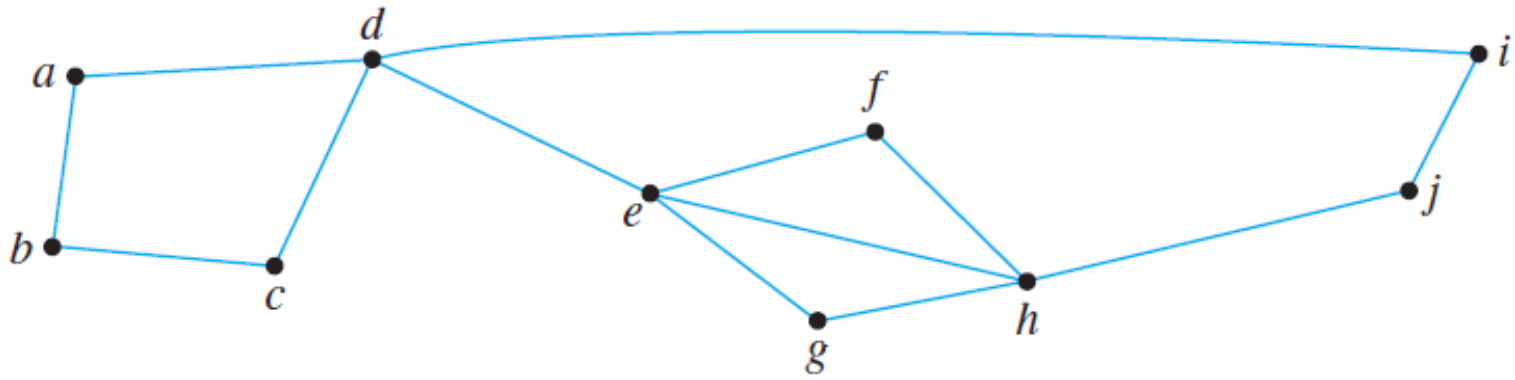
Step 3e: Let $C = C''$ and go back to step 3.

Since the graph G is finite, execution of the steps outlined in this algorithm must eventually terminate. At that point an Euler circuit for G will have been constructed. (Note that because of the element of choice in steps 1, 2, 3b, and 3c, a variety of different Euler circuits can be produced by using this algorithm.)

Example:

Finding an Euler Circuit

Use Theorem 10.2.3 to check that the graph below has an Euler circuit. Then use the algorithm from the proof of the theorem to find an Euler circuit for the graph.



Solution Observe that

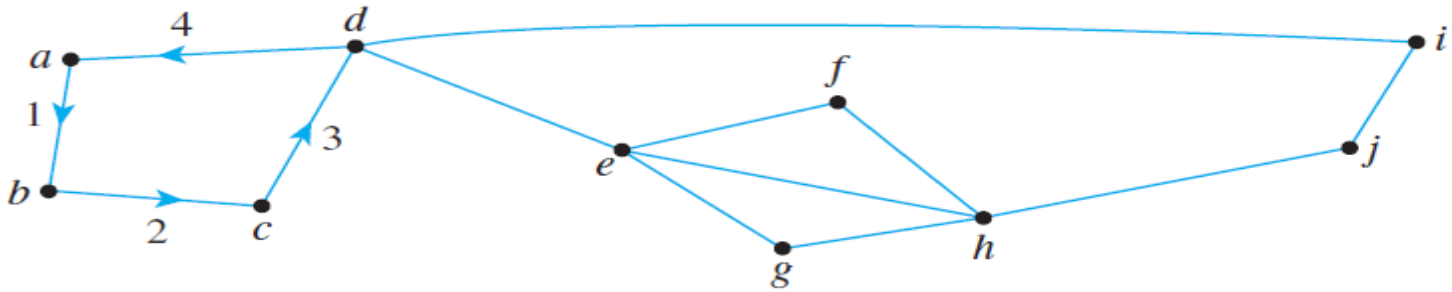
$$\deg(a) = \deg(b) = \deg(c) = \deg(f) = \deg(g) = \deg(i) = \deg(j) = 2$$

and that $\deg(d) = \deg(e) = \deg(h) = 4$. Hence all vertices have even degree. Also, the graph is connected. Thus, by Theorem 10.2.3, the graph has an Euler circuit.

To construct an Euler circuit using the algorithm of Theorem 10.2.3, let $v = a$ and let C be

$$C: abcda.$$

C is represented by the labeled edges shown below.



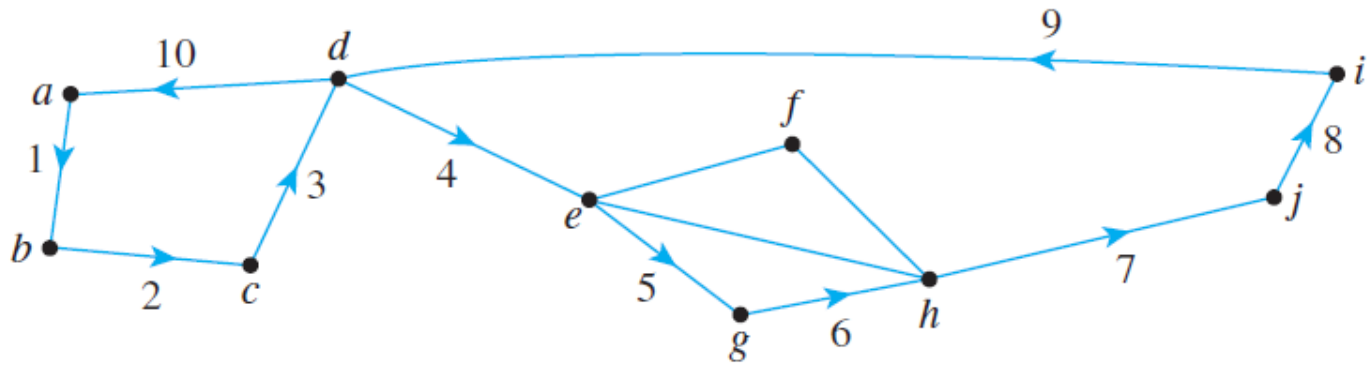
Observe that C is not an Euler circuit for the graph but that C intersects the rest of the graph at d. Let C' be

$$C': deghjid.$$

Patch C' into C to obtain

$$C'': abcdeghjida.$$

Set $C = C''$. Then C is represented by the labeled edges shown below.



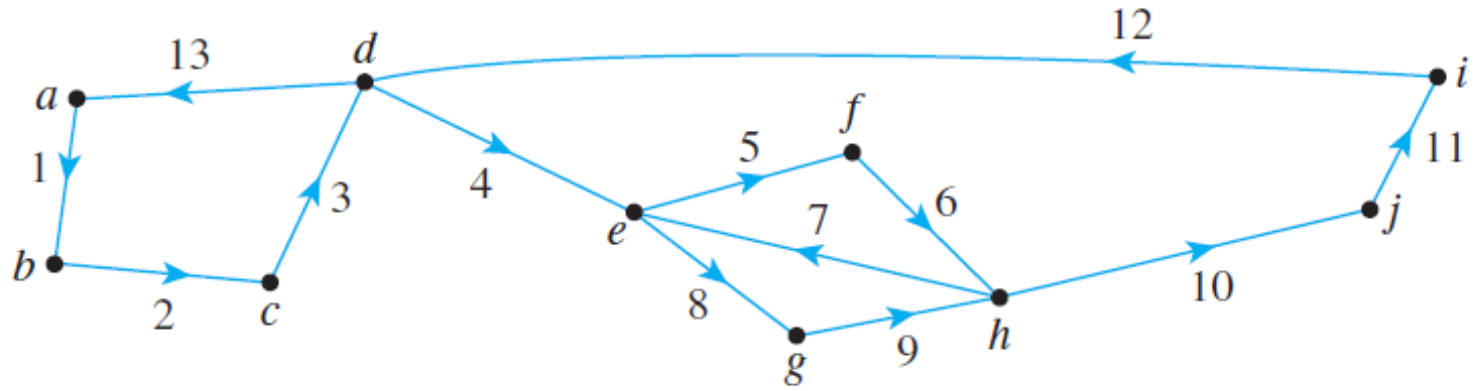
Observe that C is not an Euler circuit for the graph but that it intersects the rest of the graph at e. Let C' be

$$C'\colon efhe.$$

Patch C' into C to obtain

$$C''\colon abcdefheghjida.$$

Set $C = C''$. Then C is represented by the labeled edges shown below.



Since C includes every edge of the graph exactly once, C is an Euler circuit for the graph.

Searching Using Graphs

Breadth First Search (BFS) Algorithm

Given $G(V, E)$ represented in terms of adjacency lists A_v and a distinguished *source* vertex $s \in V$, find $d(v) \equiv d(s, v)$ for all $v \in V$.

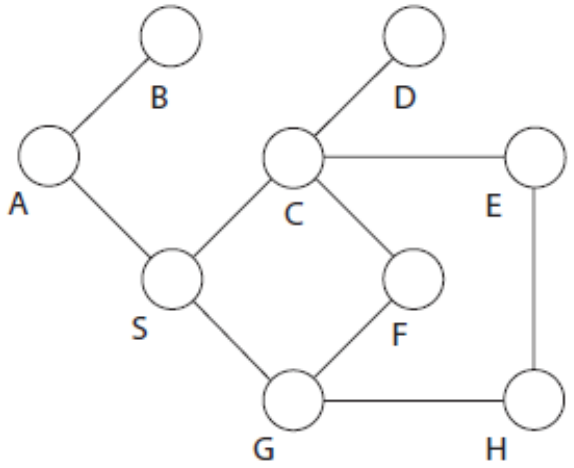
1. *Set things up:*
   $d(v) \leftarrow \infty$ for all $v \neq s \in V$
   $d(s) \leftarrow 0$ and $Q \leftarrow \{s\}$

2. *Main loop: continues until the queue is empty*

```
While ( Q ≠ ∅ ) {
        Pop a vertex v off the left end of Q.

        Examine each of v's neighbours
        For each  w ∈ A_v {
            If(  d(w) = ∞  ) then {
                Set d(w) and get ready to process w's neighbours
                d(w) ← d(v) + 1
                Push w on to the right end of Q.
            }
        }
}
```

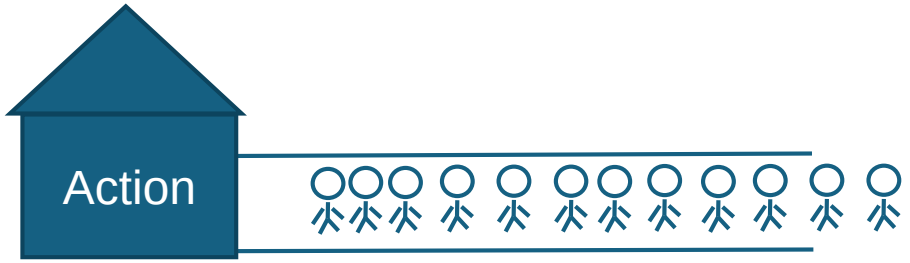
Example:



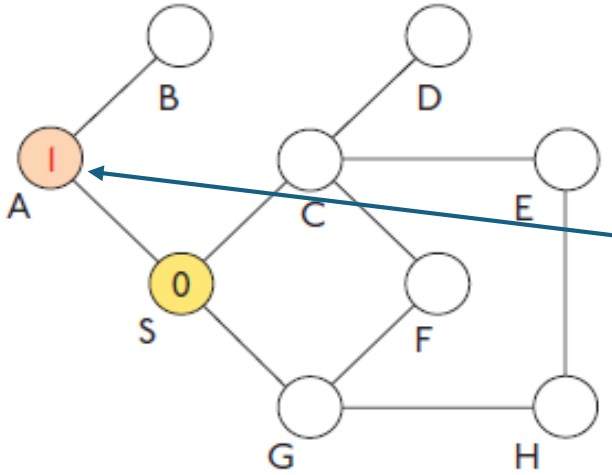
Take S to be the source vertex.

Adjacancy Lists:

$$
\begin{aligned}
A_S &= \{A, C, G\} \\
A_A &= \{B, S\} \\
A_B &= \{A\} \\
A_C &= \{D, E, F, S\} \\
A_D &= \{C\} \\
A_E &= \{C, H\} \\
A_F &= \{C, G\} \\
A_G &= \{F, H, S\} \\
A_H &= \{E, G\}
\end{aligned}
$$



Breadth First Search: in action



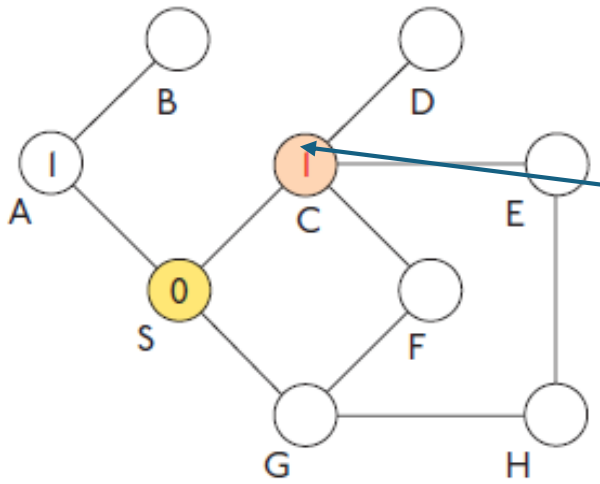
v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$

$d_a = d_s + 1$

Yellow vertex is v, red is w.

$$A_v = A_S = \{A, C, G\}$$

Breadth First Search: in action



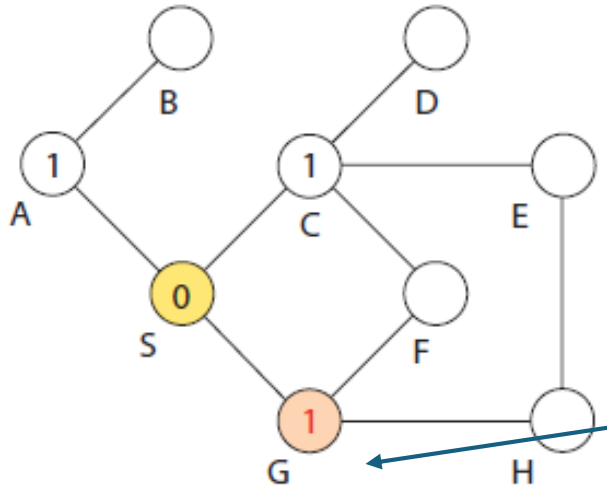
v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$

$$d_c = d_s + 1$$

Yellow vertex is v, red is w.

$$A_v = A_S = \{A, C, G\}$$

Breadth First Search: in action



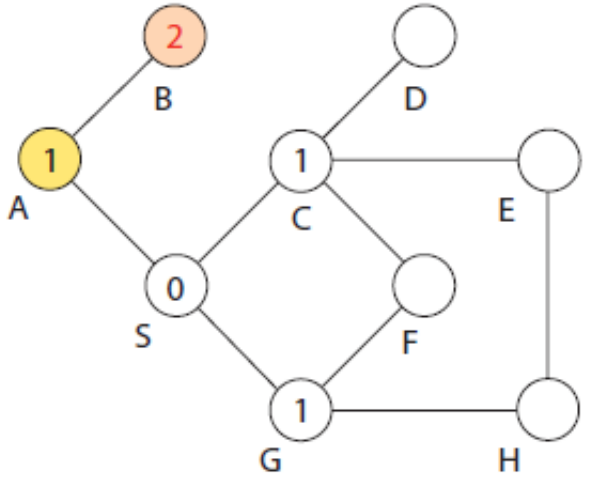
v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$

$d_G = d_s + 1$

Yellow vertex is v, red is w.

$$A_v = A_S = \{A, C, G\}$$

Breadth First Search: in action

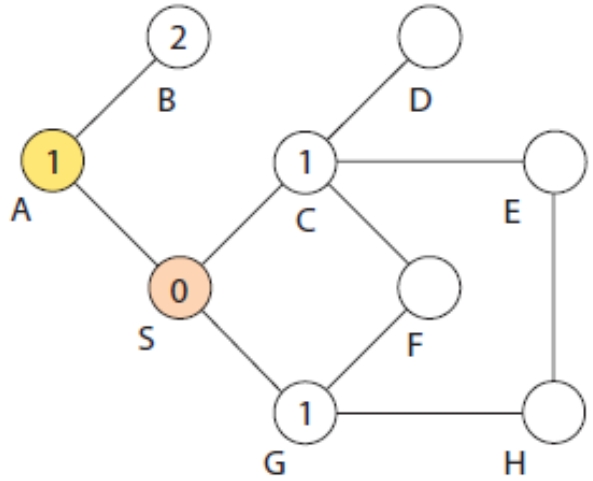


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$

Yellow vertex is v, red is w.

$$A_v = A_A = \{B, S\}$$

Breadth First Search: in action

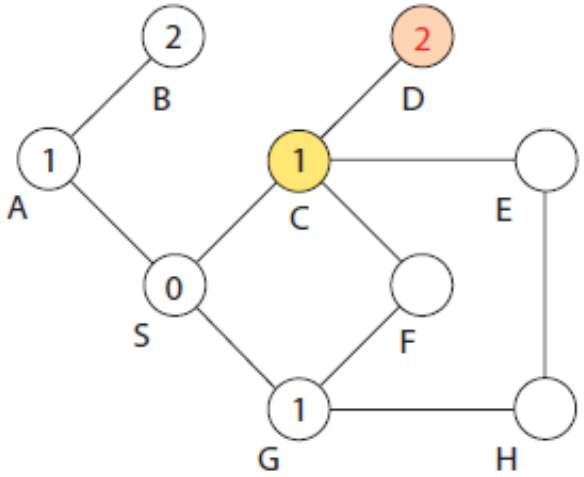


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$

Yellow vertex is v, red is w.

$$A_v = A_A = \{B, S\}$$

Breadth First Search: in action

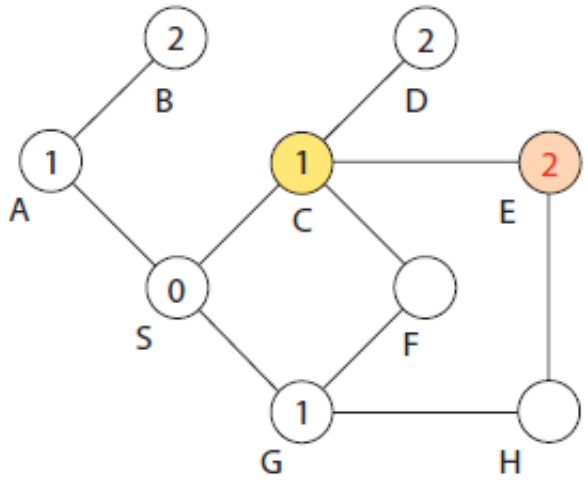


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$

Yellow vertex is v, red is w.

$$A_v = A_C = \{D, E, F, S\}$$

Breadth First Search: in action

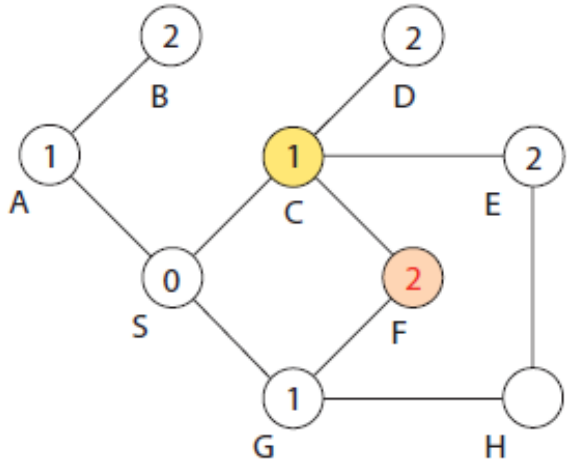


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$

Yellow vertex is v, red is w.

$$A_v = A_C = \{D, E, F, S\}$$

Breadth First Search: in action

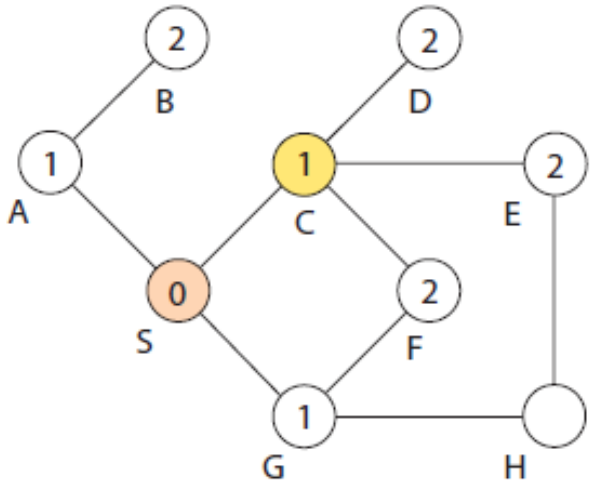


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$

Yellow vertex is v, red is w.

$$A_v = A_C = \{D, E, F, S\}$$

Breadth First Search: in action

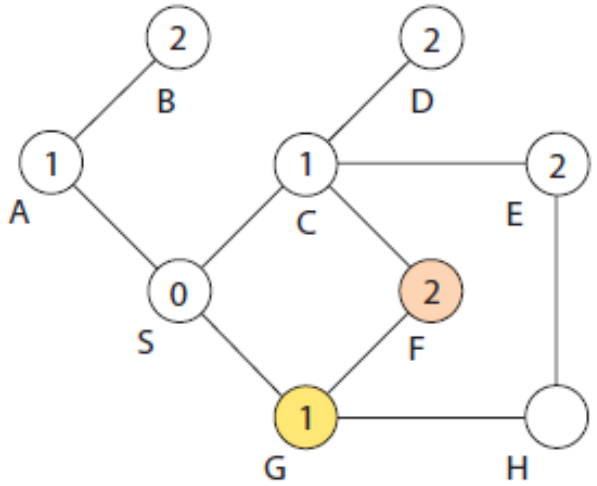


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$

Yellow vertex is v, red is w.

$$A_v = A_C = \{D, E, F, S\}$$

Breadth First Search: in action

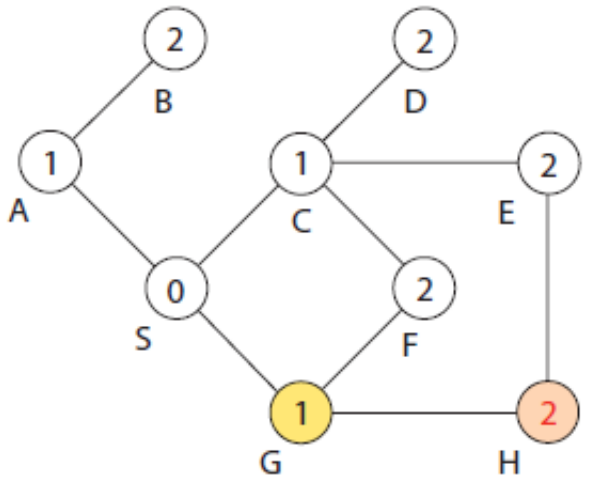


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$

Yellow vertex is v, red is w.

$$A_v = A_G = \{F, H, S\}$$

Breadth First Search: in action

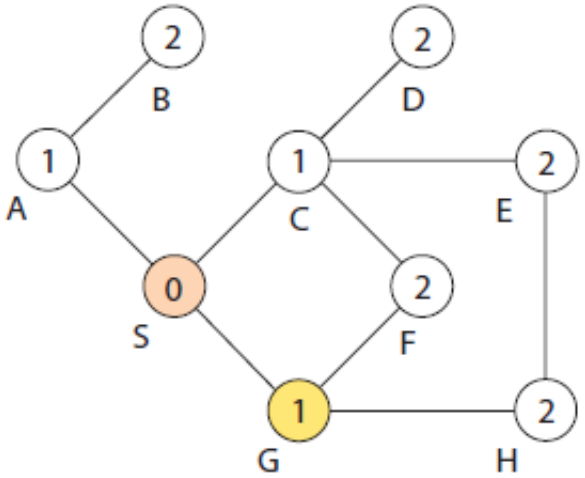


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$

Yellow vertex is v, red is w.

$$A_v = A_G = \{F, H, S\}$$

Breadth First Search: in action

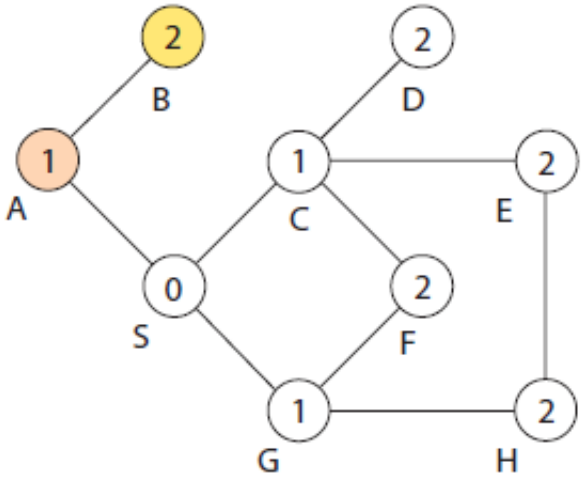


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$

Yellow vertex is v, red is w.

$$A_v = A_G = \{F, H, S\}$$

Breadth First Search: in action

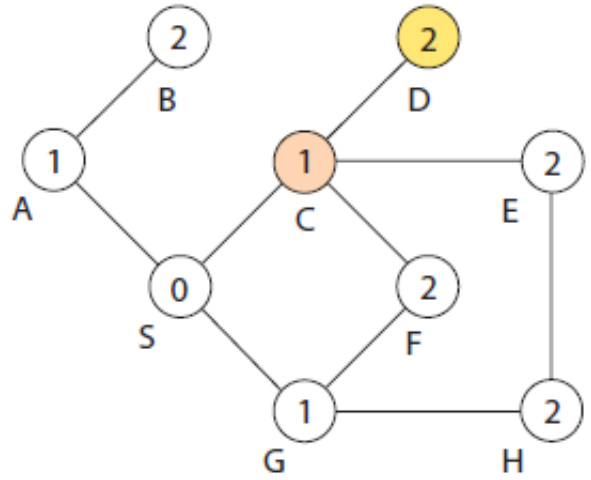


Yellow vertex is v, red is w.

$$A_v = A_B = \{A\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$

Breadth First Search: in action

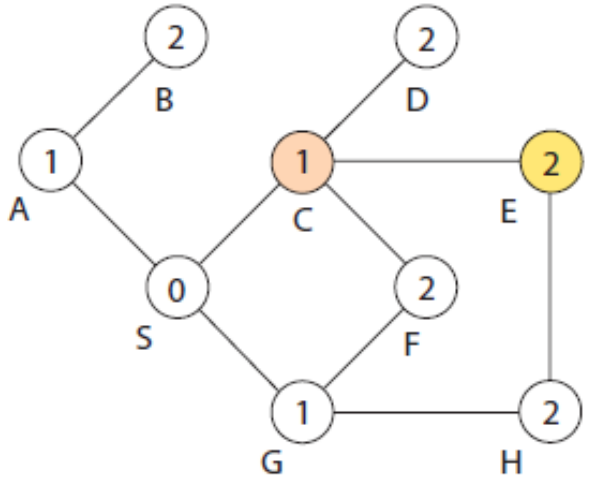


Yellow vertex is v, red is w.

$$A_v = A_D = \{C\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$

Breadth First Search: in action

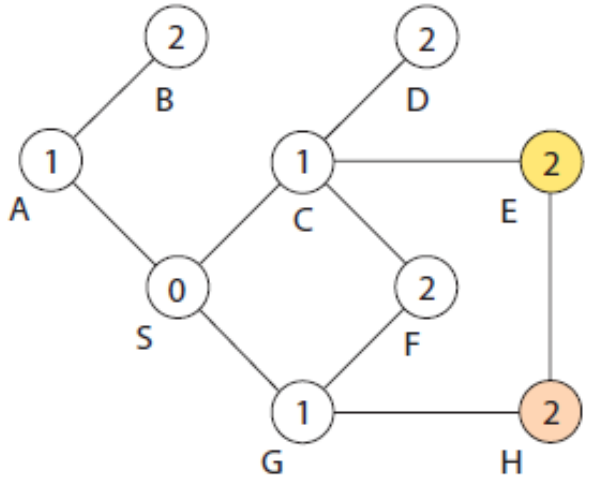


Yellow vertex is v, red is w.

$$A_v = A_E = \{C, H\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$

Breadth First Search: in action

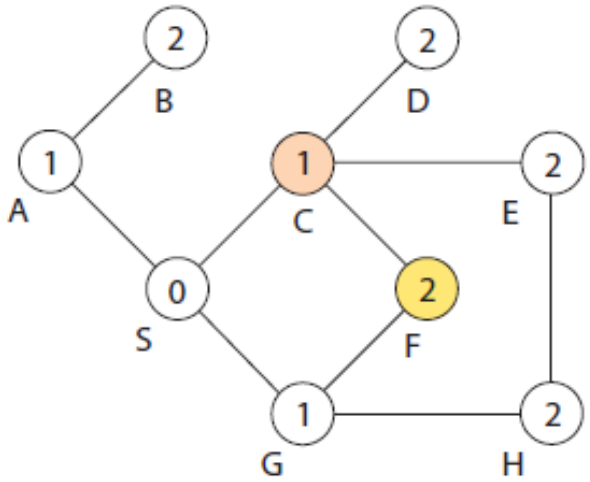


Yellow vertex is v, red is w.

$$A_v = A_E = \{C, H\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$

Breadth First Search: in action

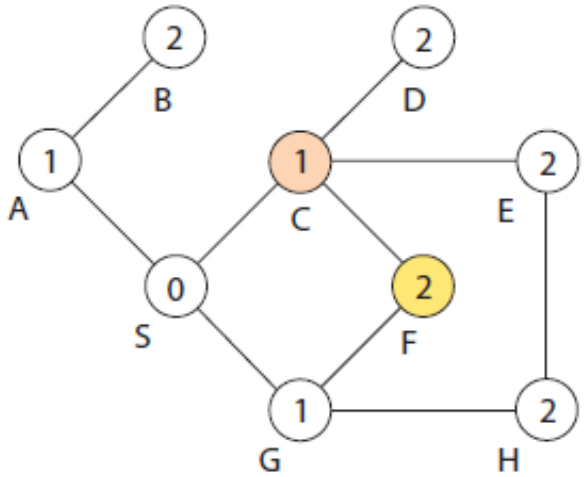


Yellow vertex is v, red is w.

$$A_v = A_F = \{C, G\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$

Breadth First Search: in action

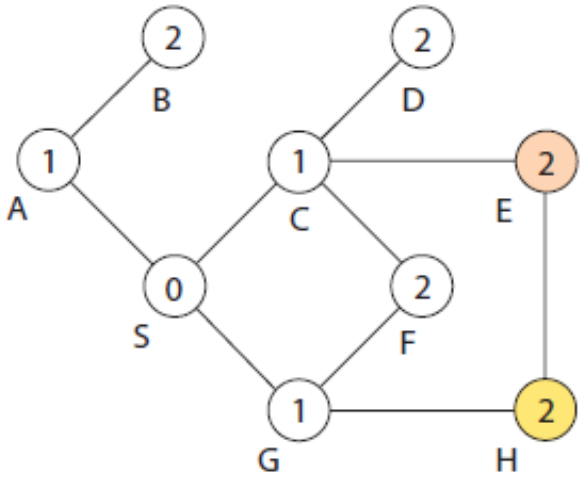


Yellow vertex is v, red is w.

$$A_v = A_F = \{C, G\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$

Breadth First Search: in action

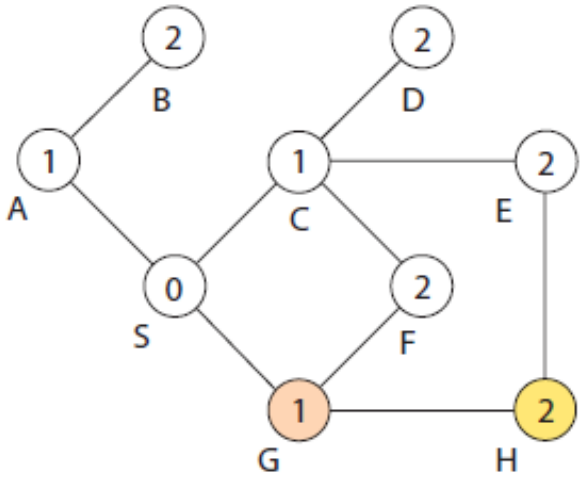


Yellow vertex is v, red is w.

$$A_v = A_H = \{E, G\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$
F	G	none	$\{H\}$
H	E	none	$\{\}$

Breadth First Search: in action

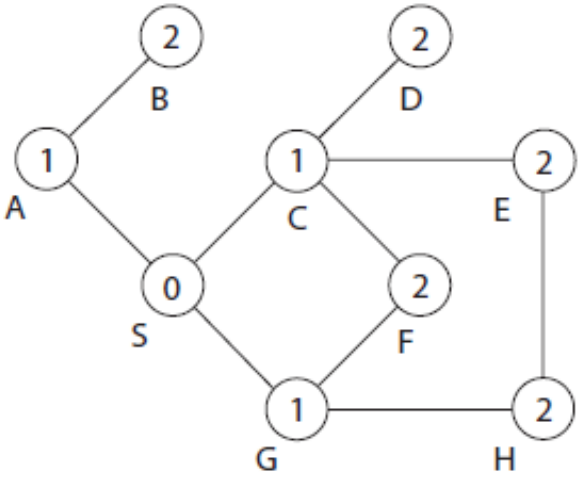


Yellow vertex is v, red is w.

$$A_v = A_H = \{E, G\}$$

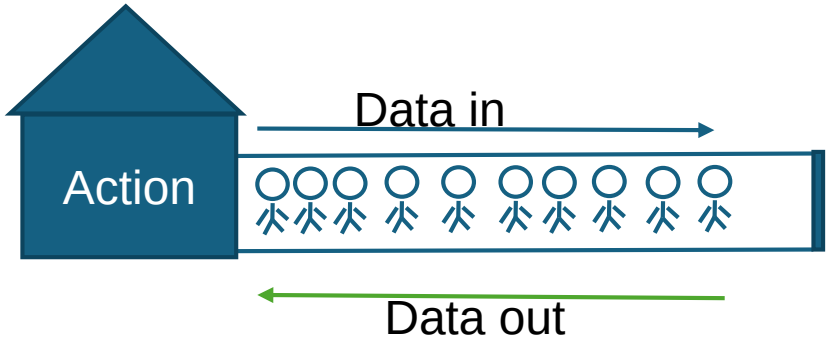
v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$
F	G	none	$\{H\}$
H	E	none	$\{\}$
H	G	none	$\{\}$

Breadth First Search: in action

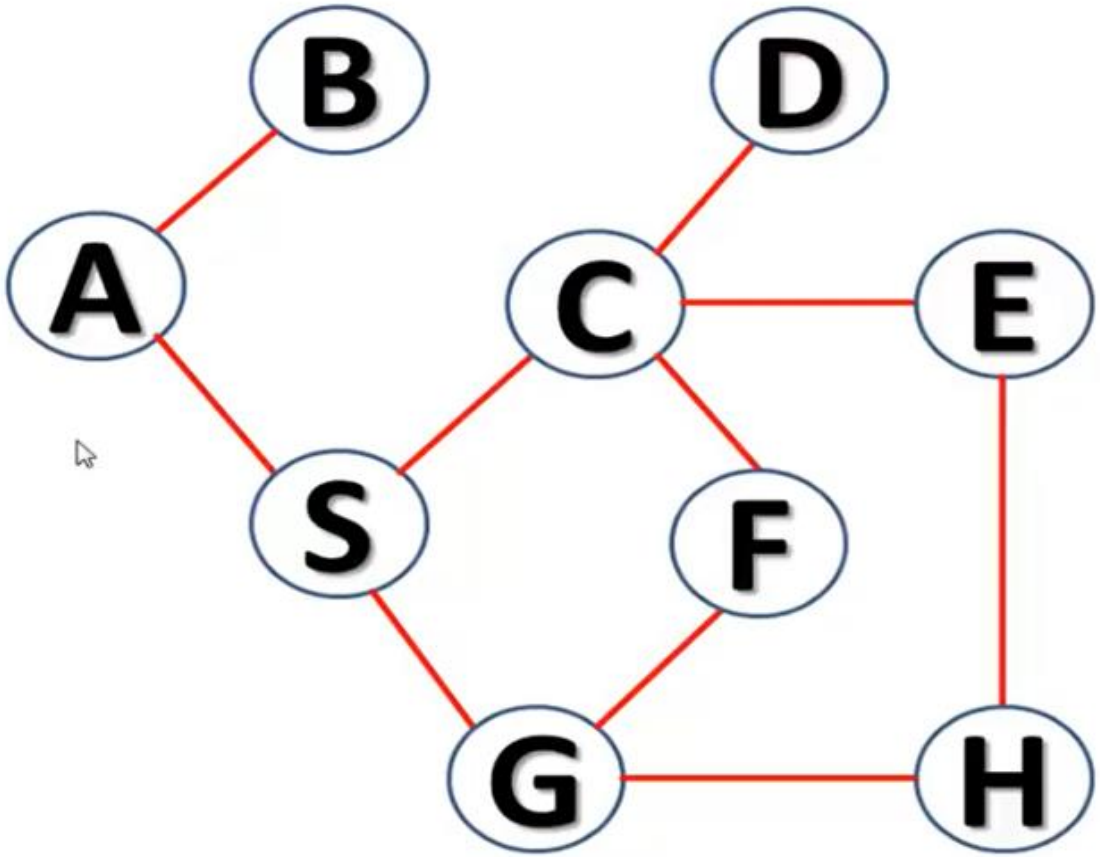


v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$
F	G	none	$\{H\}$
H	E	none	$\{\}$
H	G	none	$\{\}$

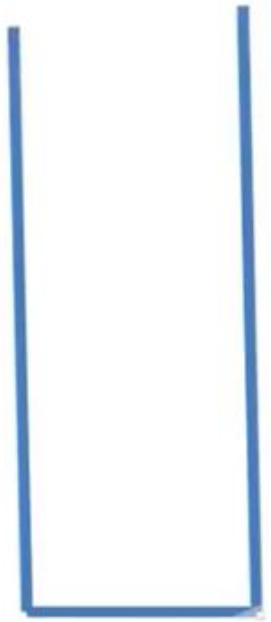
Depth First Search (DFS)



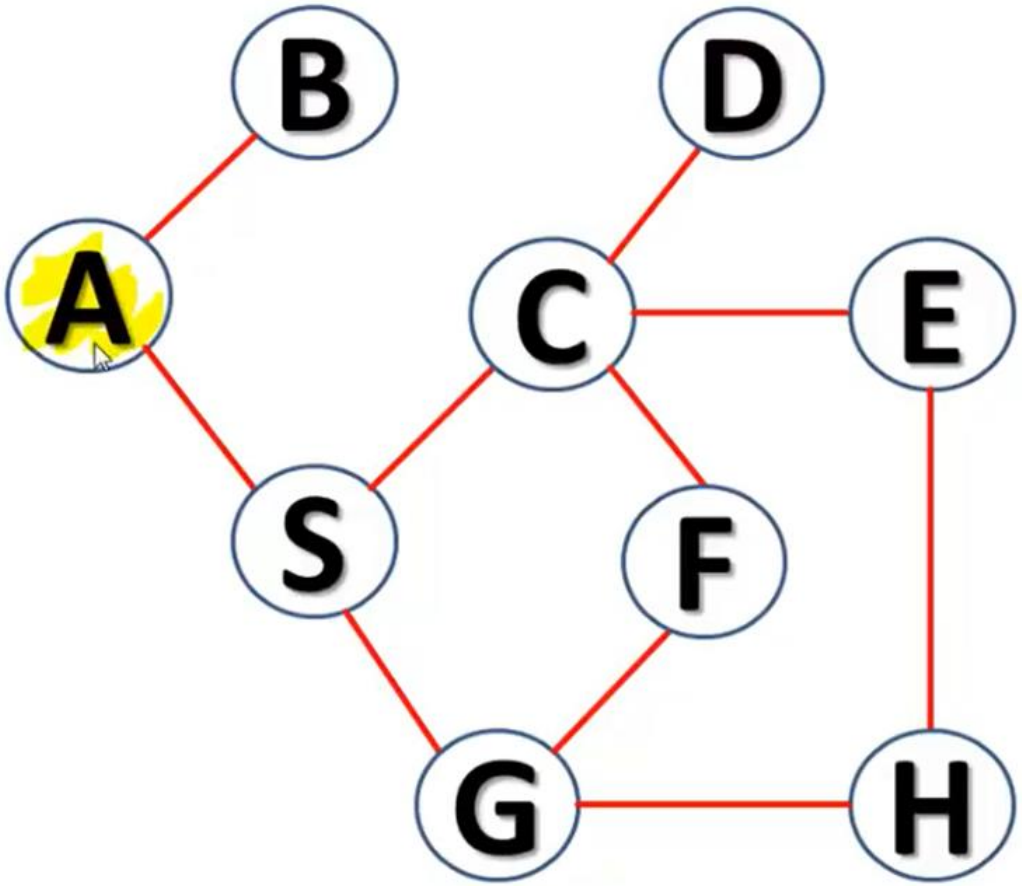
Depth First Search (DFS)



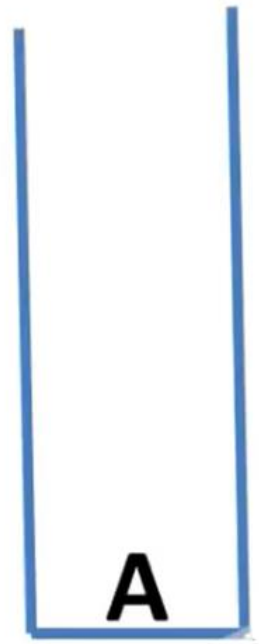
Stack Status



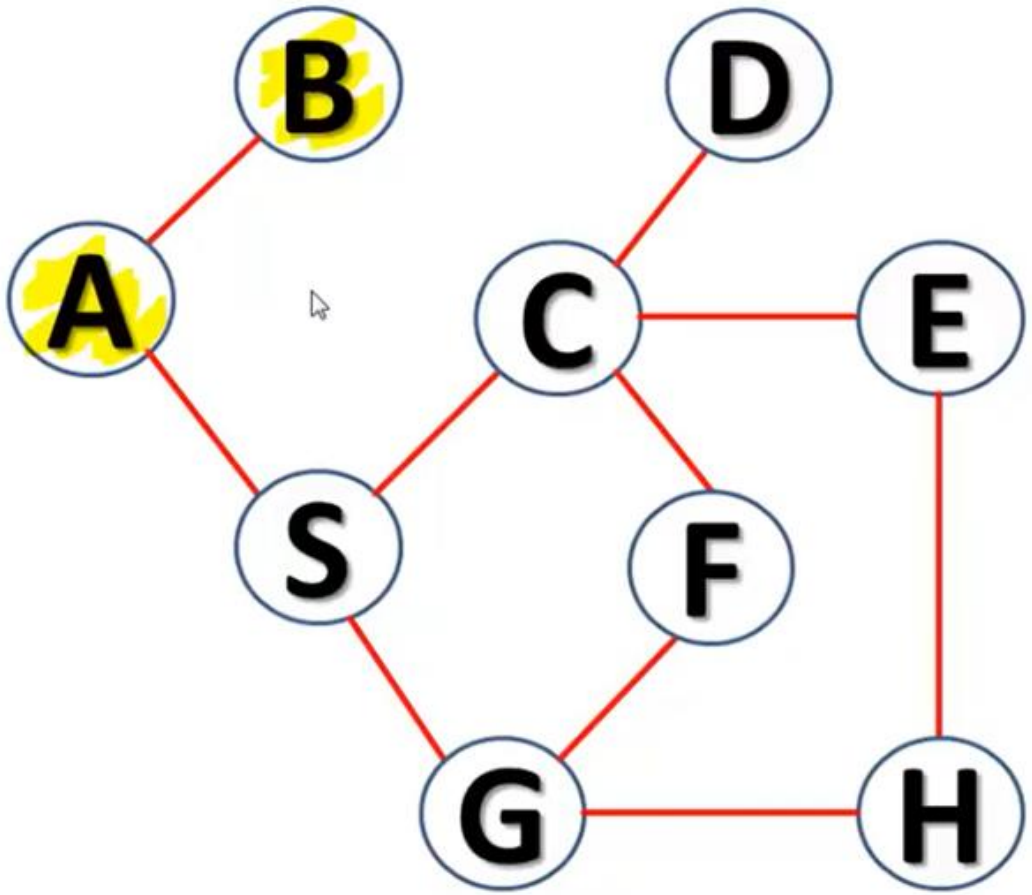
OUTPUT :



Stack Status

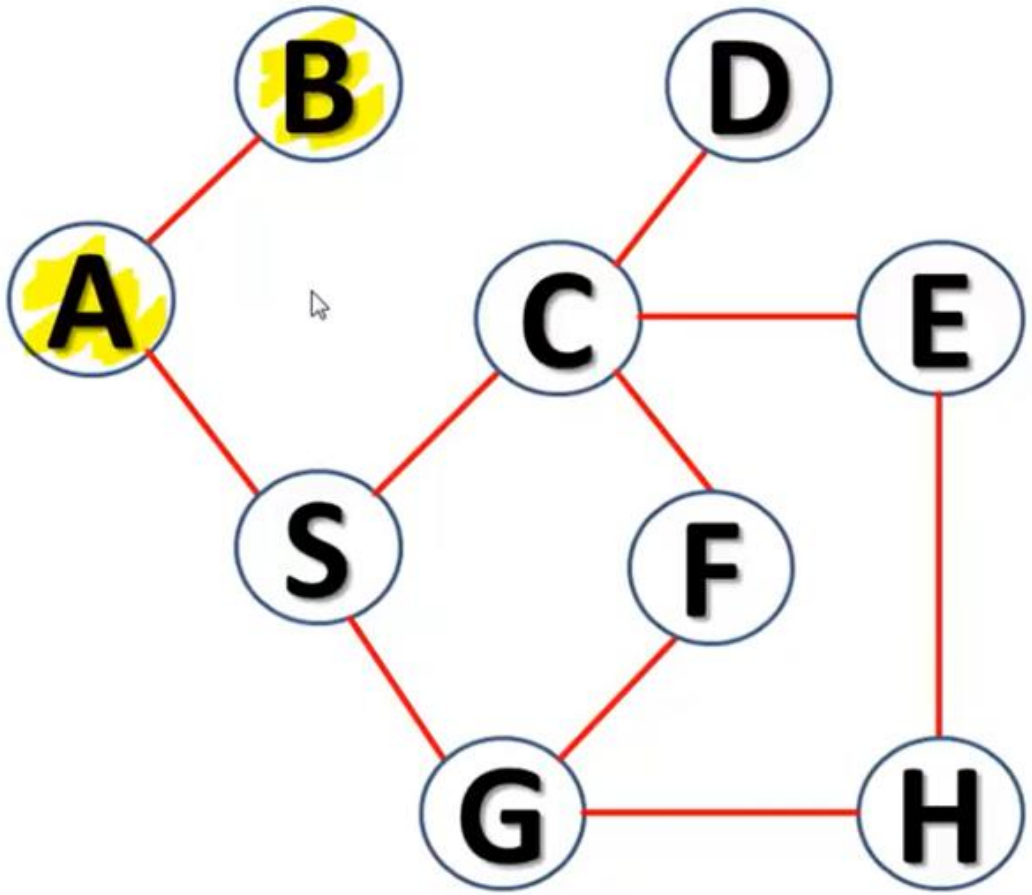


OUTPUT : A

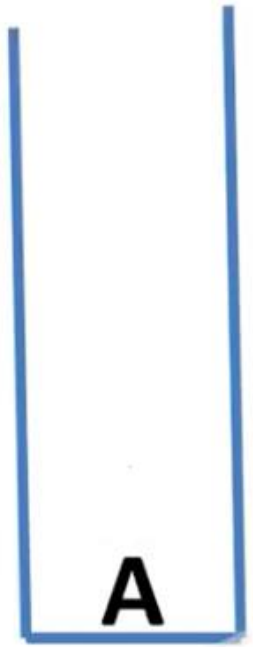


Stack Status

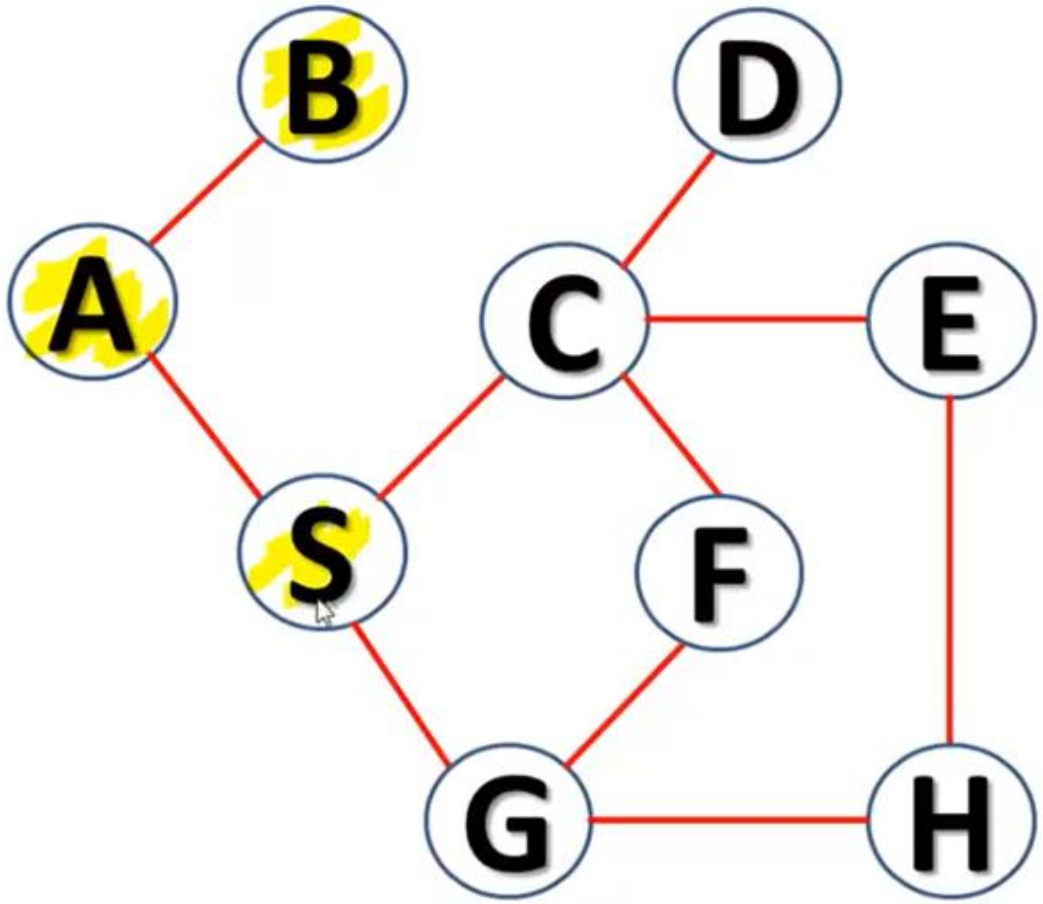
OUTPUT : A B



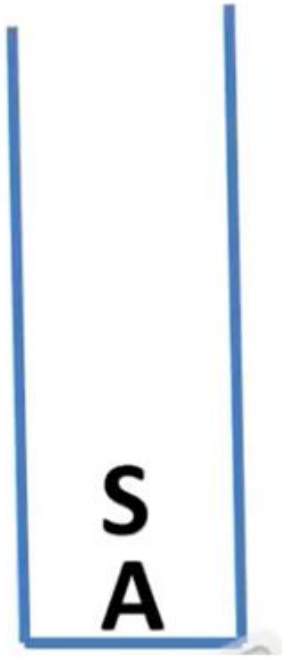
Stack Status



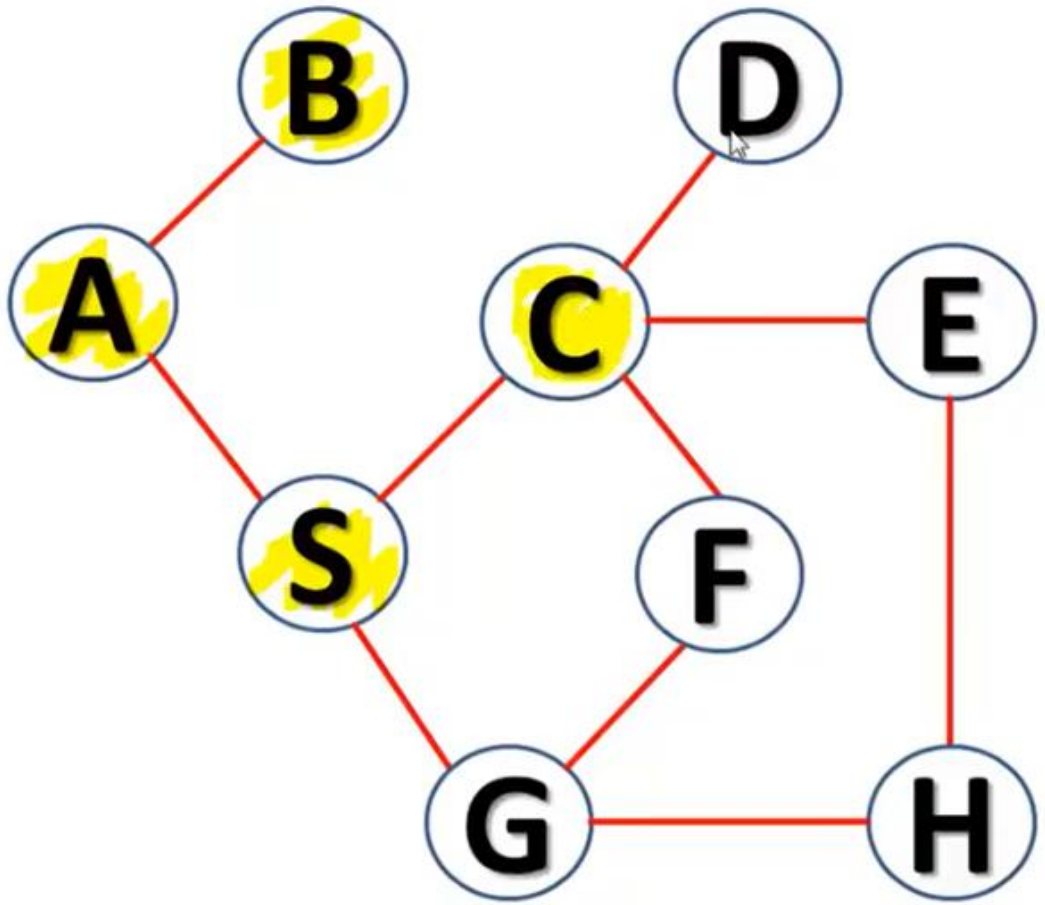
OUTPUT : A B



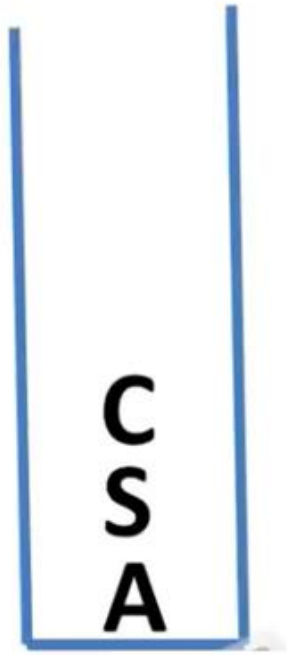
Stack Status



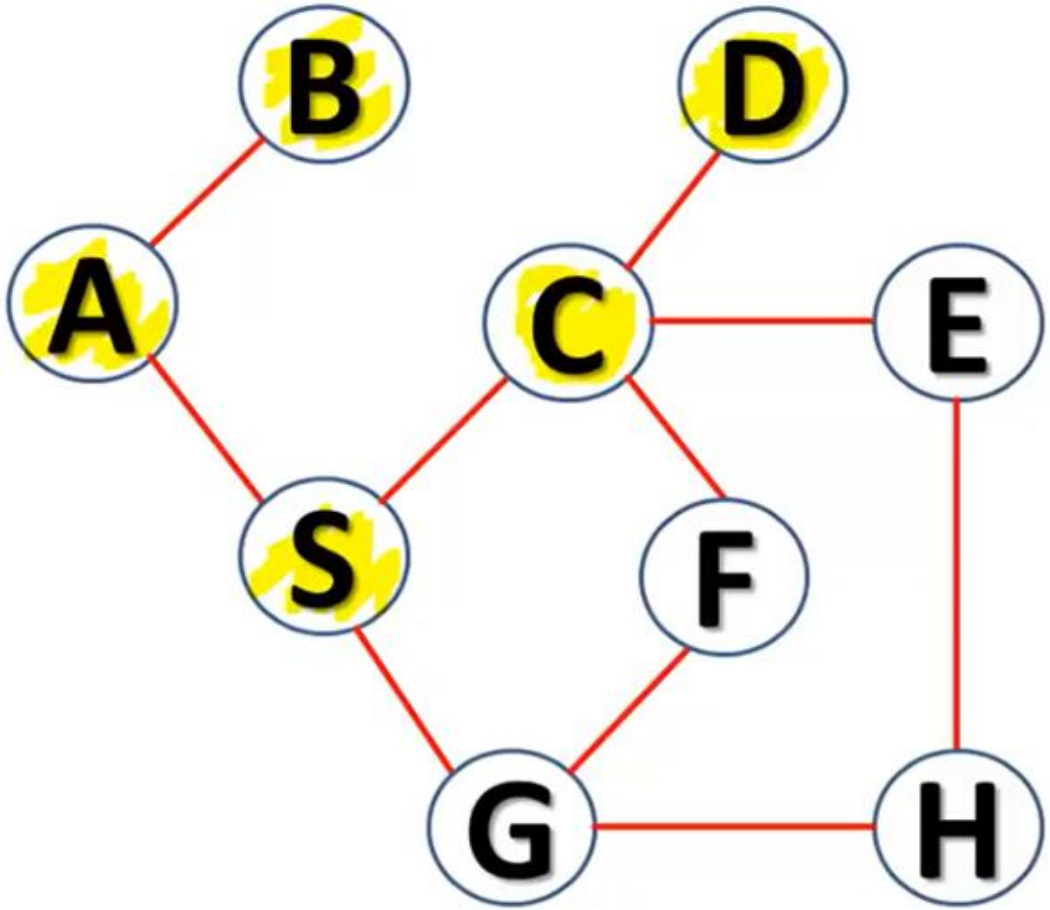
OUTPUT : A B S



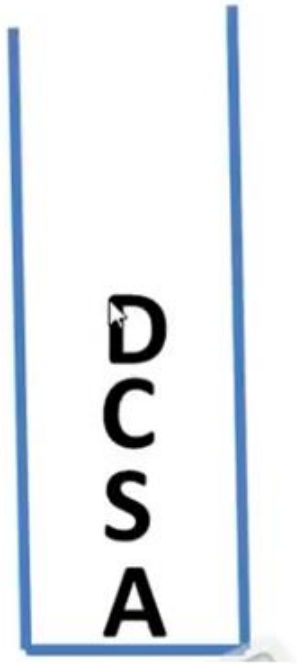
Stack Status



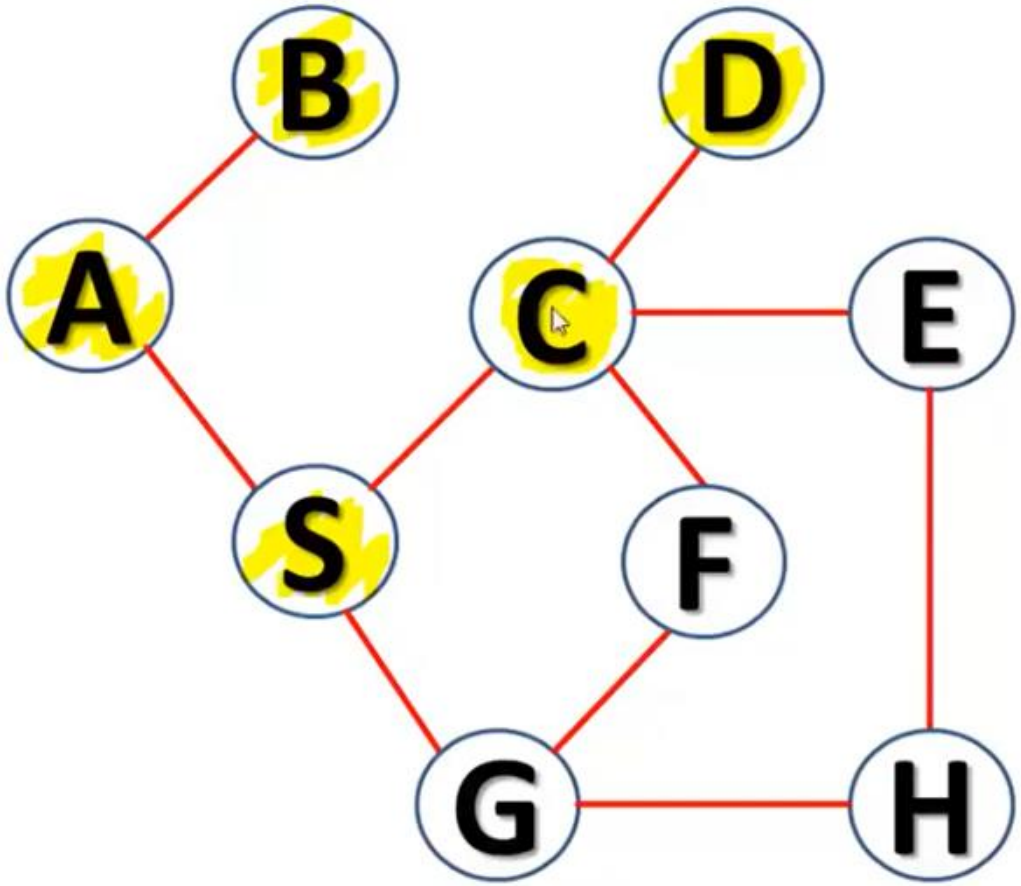
OUTPUT : A B S C



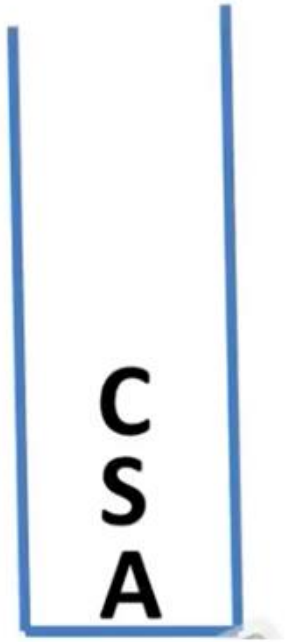
Stack Status



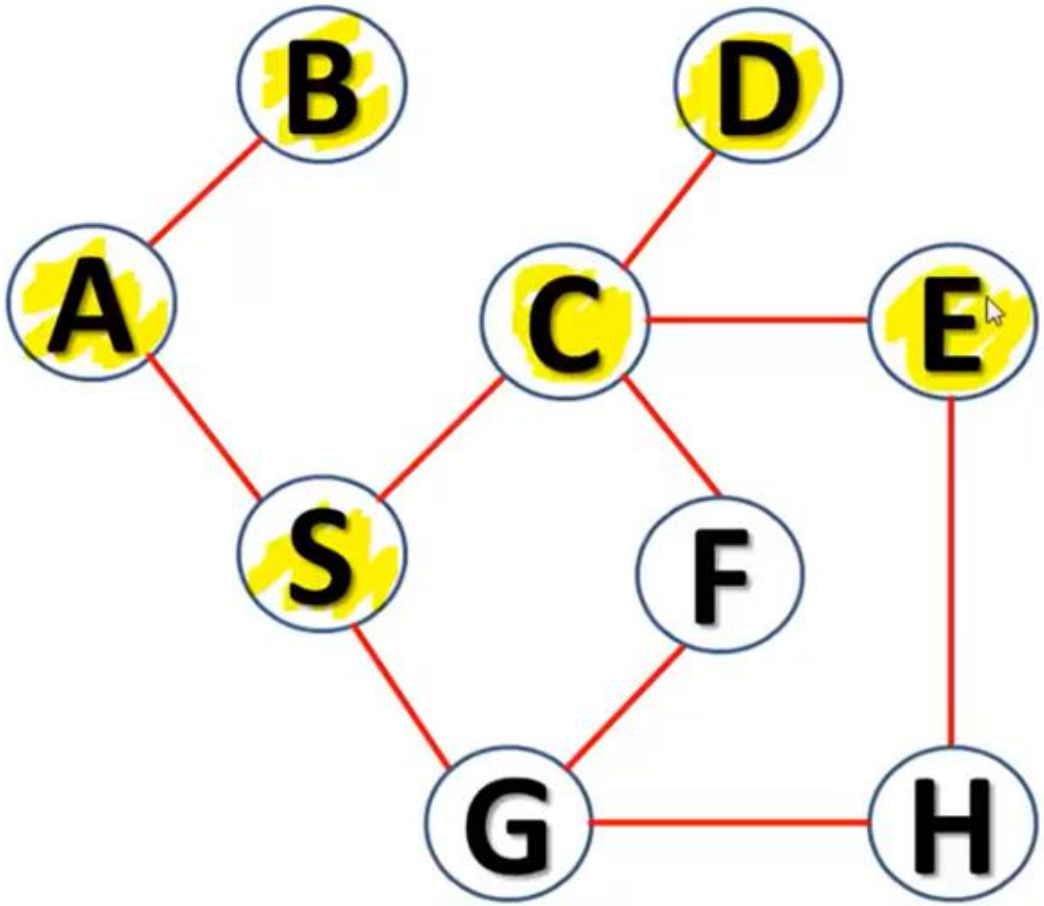
OUTPUT : A B S C



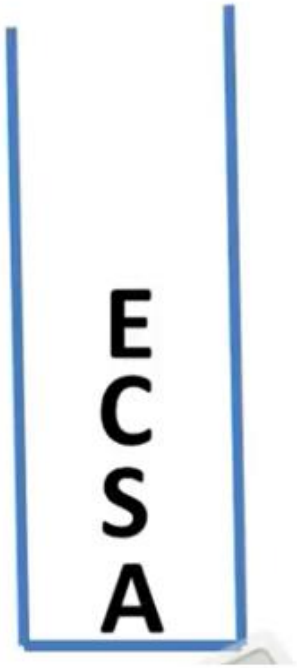
Stack Status



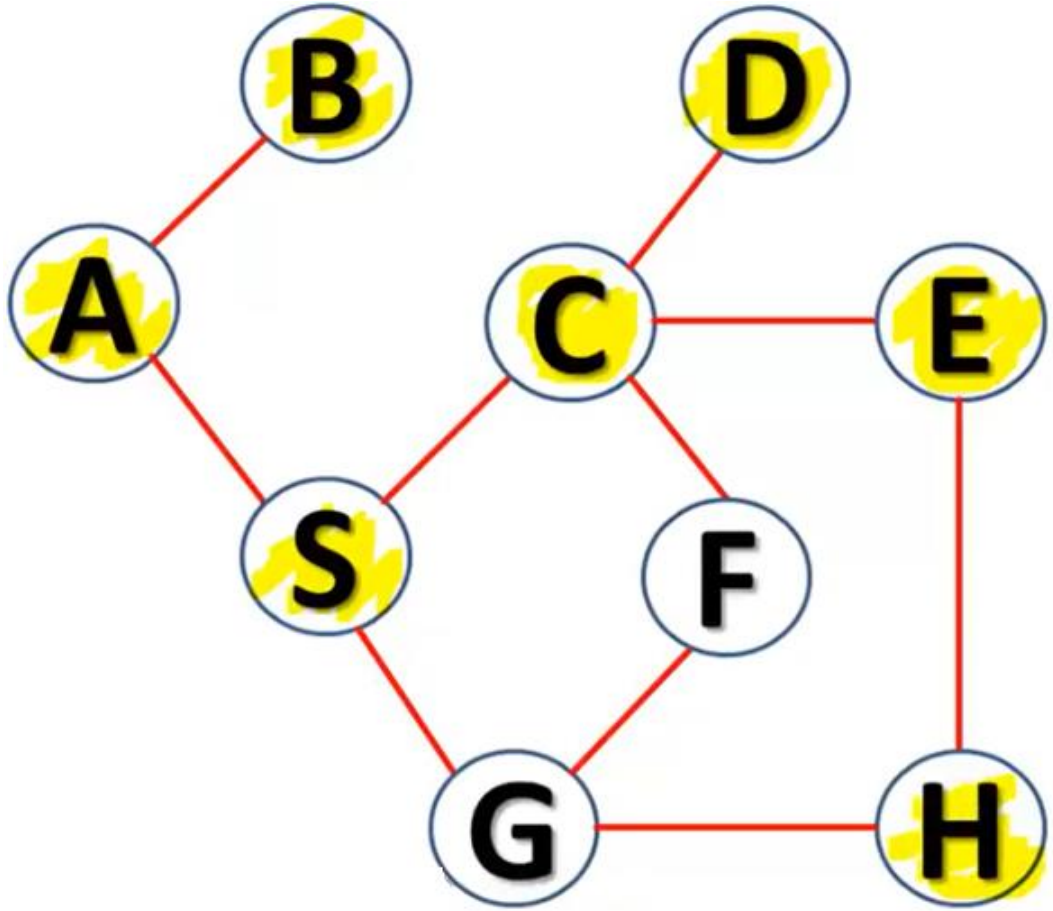
OUTPUT : A B S C D



Stack Status

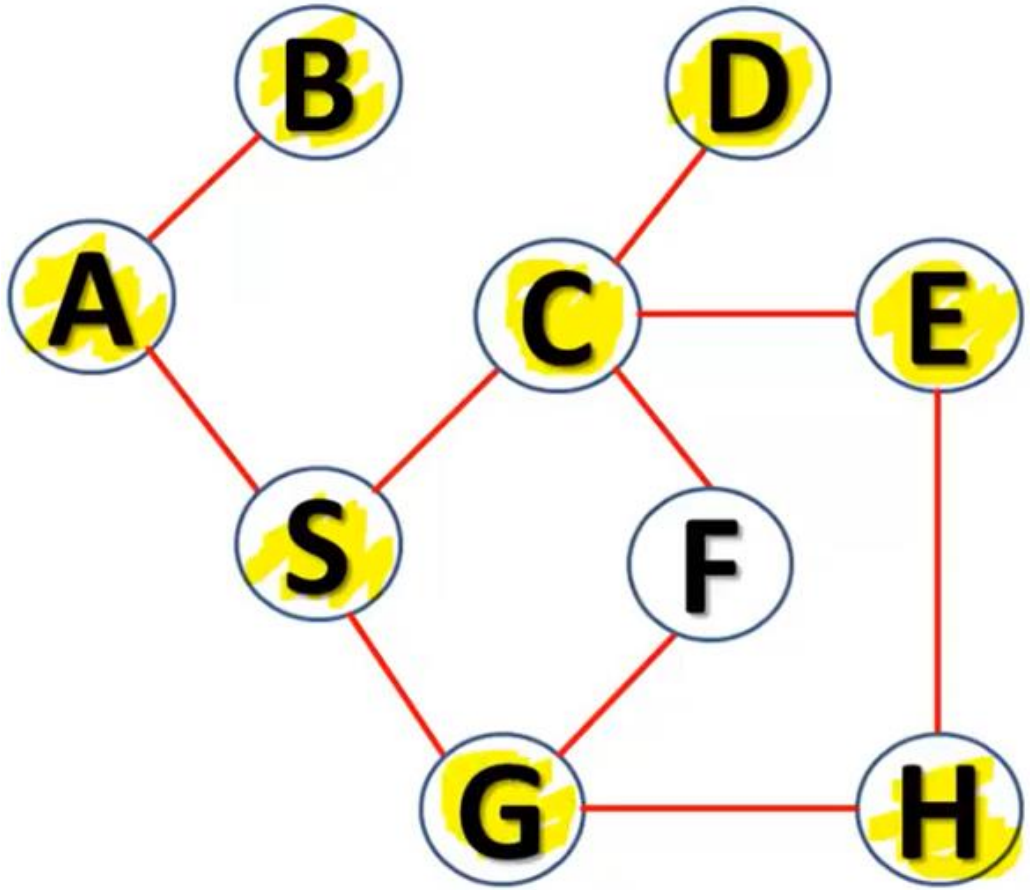


OUTPUT : A B S C D E



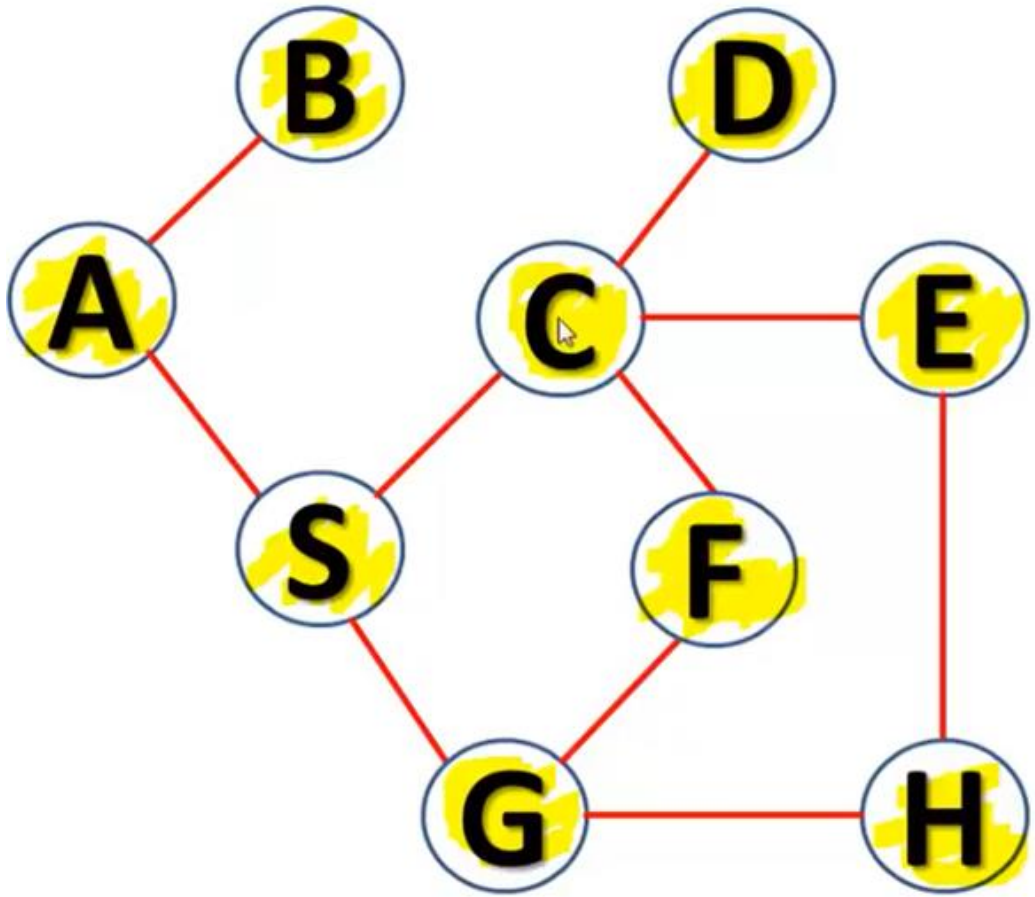
Stack Status

OUTPUT : A B S C D E H



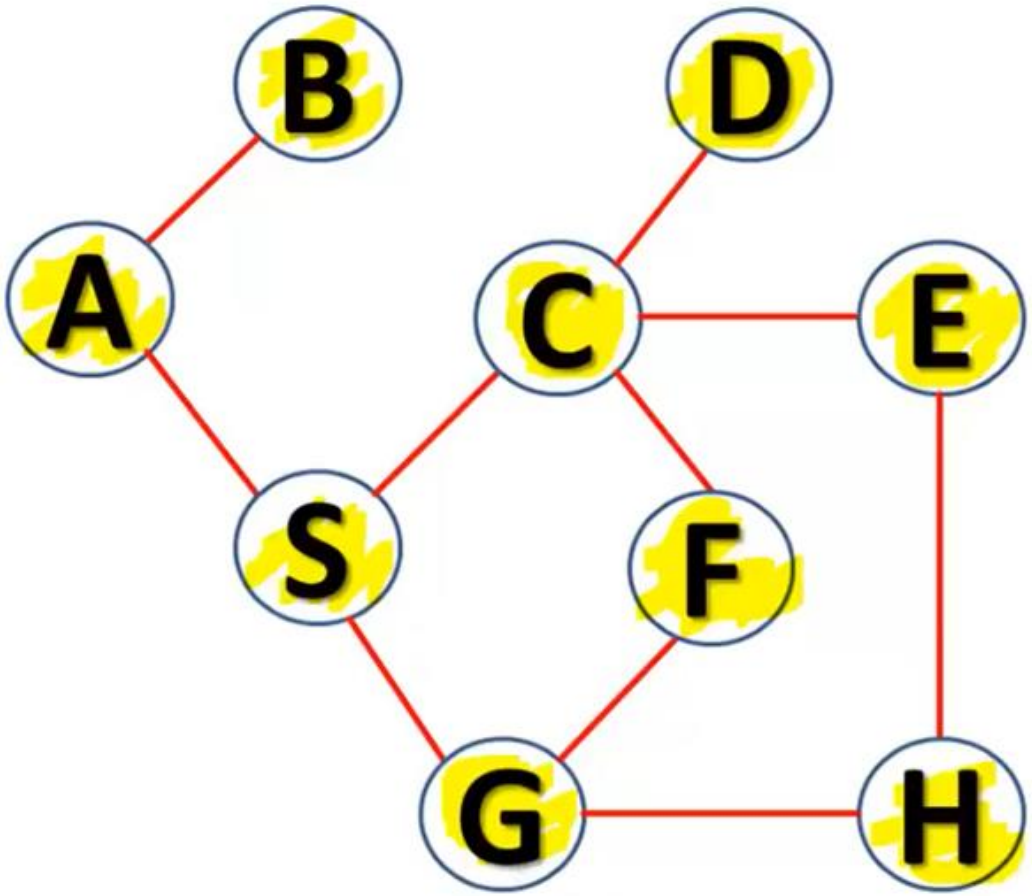
Stack Status

OUTPUT : A B S C D E H G



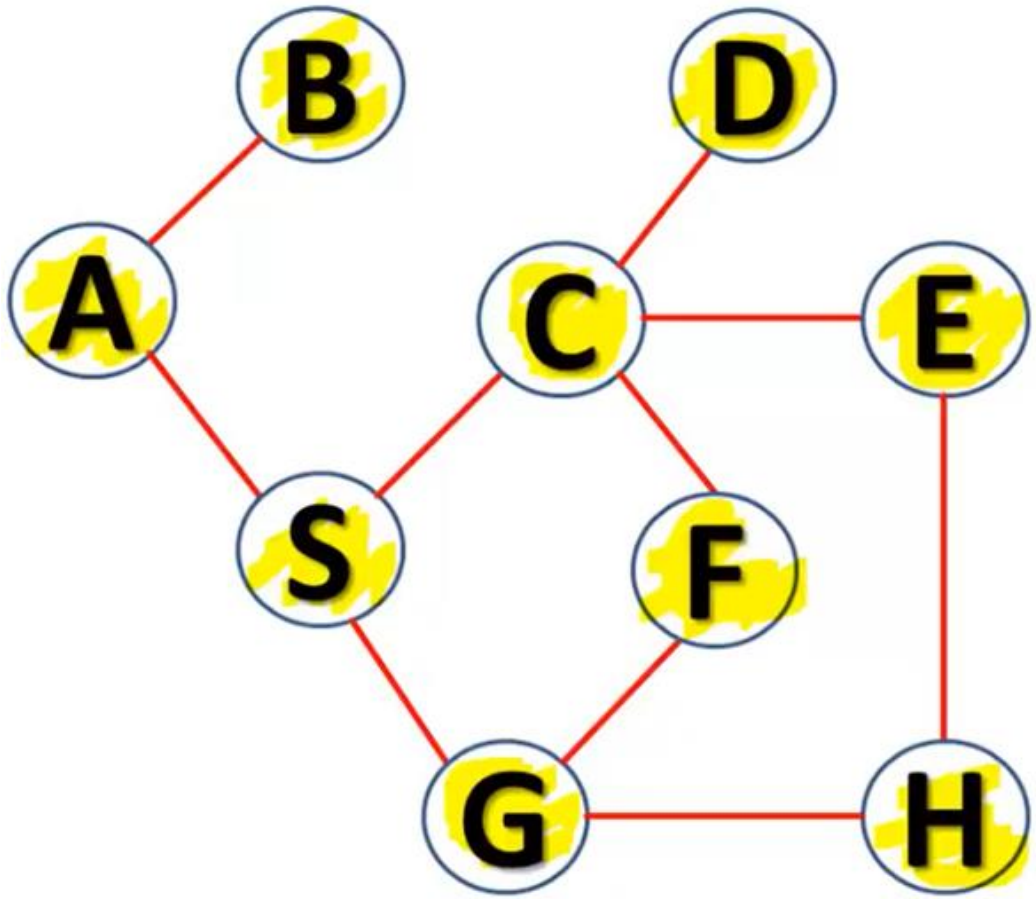
Stack Status

OUTPUT : A B S C D E H G F



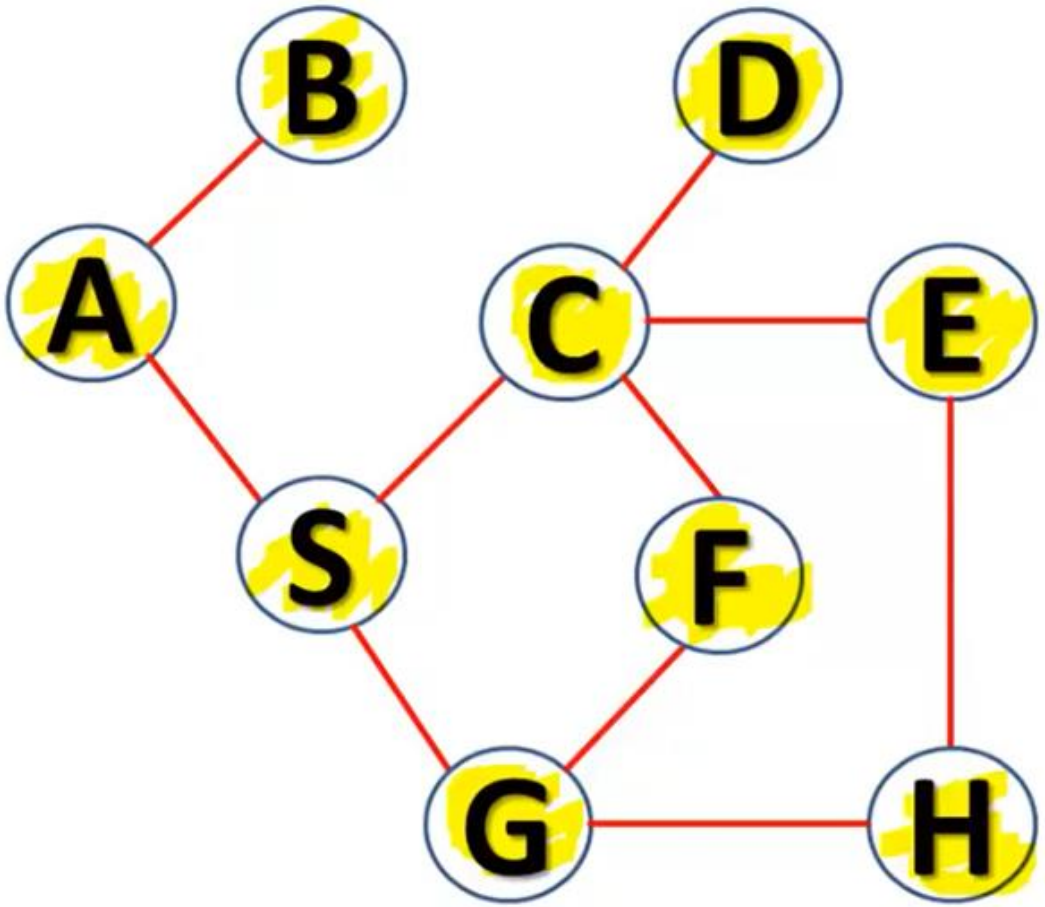
Stack Status

OUTPUT : A B S C D E H G F



Stack Status

OUTPUT : A B S C D E H G F



Stack Status

Stack Empty

OUTPUT : A B S C D E H G F

Pros and Cons of BFS and DFS

- DFS - pros: efficient in space, as it only has to remember the current branch
- DFS - cons: could loop forever, may not find a good path
- BFS: - automatically finds the “shortest” path, where path length is number of edges
- BFS - cons - very inefficient in space in dense graphs, as the queue can grow long - essentially, for path finding you are maintaining all possible paths