Graph

Today’s Lecture

- Graph Definition
- Graph Terminology
- Representation of Graph
- Path and Cycle
- Graph Traversal (BFS-DFS)

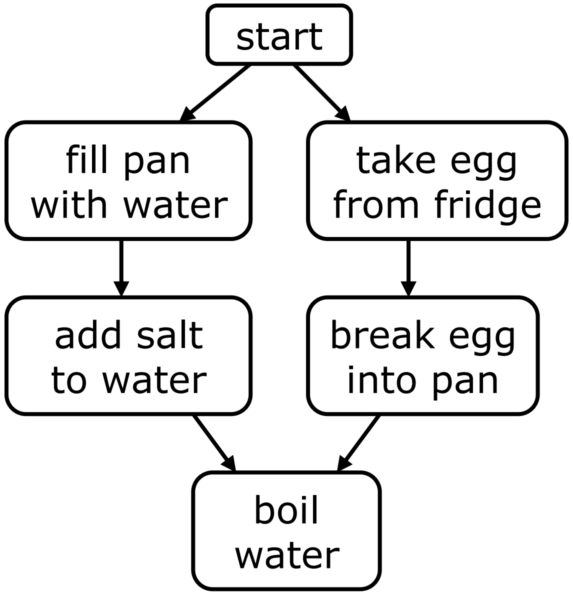
Graph definitions

- A **graph** is a collection of **nodes** (or **vertices**, singular is vertex) and **edges** (or **arcs**)
  - Each node contains an **element**
  - Each edge connects two nodes together (or possibly the same node to itself) and may contain an **edge attribute or weight**

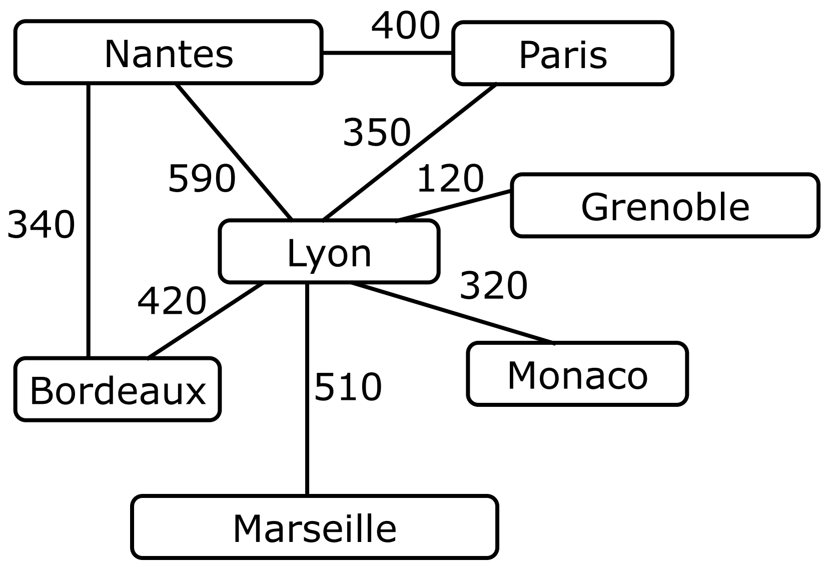
Directed And Undirected Graphs

- There are two kinds of graphs: directed graphs (sometimes called **digraphs**) and **undirected graphs**
  - A **directed graph** is one in which the edges have a direction
  - An **undirected graph** is one in which the edges do not have a direction.

Directed And Undirected Graphs



A directed graph



An undirected graph

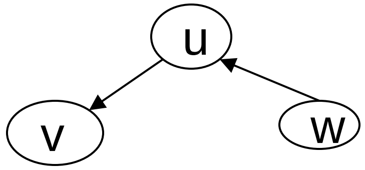
Size and Degree

- The **size** of a graph is the number of *edges* in it
- The **empty graph** has size zero (no nodes)
- If two nodes are connected by an edge, they are **neighbors** (and the nodes are **adjacent** to each other)
- The **degree of a node** is the number of edges it has

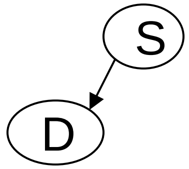
- For directed graphs,
  - In a directed graph vertex v is adjacent to u, if there is an edge leaving u and coming to v.

v is adjacent to u.

u is adjacent to w



  - If a directed edge goes from node S to node D, we call S the source and D the destination of the edge
    - The edge is an **out-edge** of S and an **in-edge** of D
    - S is a **predecessor** of D, and D is a **successor** of S

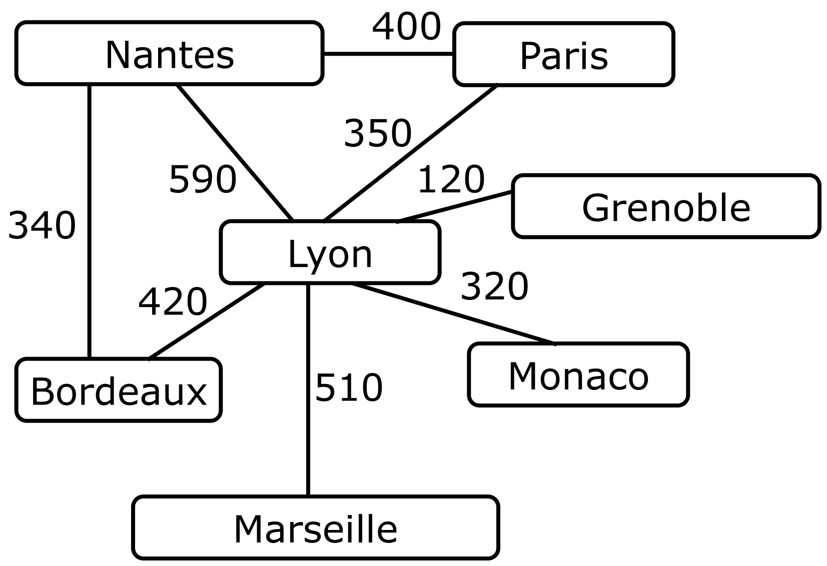


  - The **in-degree** of a node is the number of in-edges it has
  - The **out-degree** of a node is the number of out-edges it has

Path and Cycle

- In graph theory, a ***path*** in a graph is a sequence of edges which connect a sequence of vertices. Path has a first vertex, called its *start vertex*, and a last vertex, called its *end vertex*. Both of them are called *terminal vertices* of the path.

- A *cycle* is a path whose first and last nodes are the same



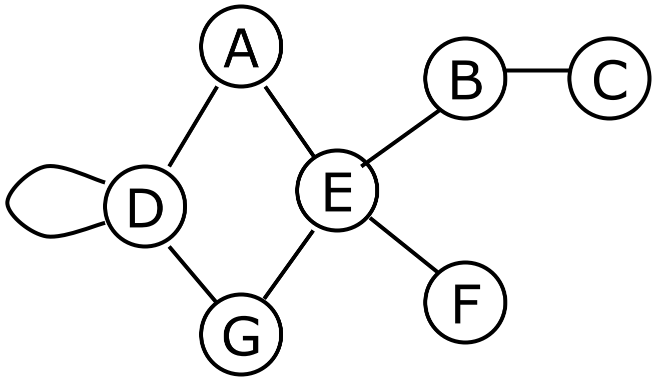
An undirected graph

- Example: (Paris, Nantes, Bordeaux, Lyon) is a path
- Example: (Paris, Nantes, Lyon, Paris) is a cycle
- A cyclic graph contains at least one cycle
- An acyclic graph does not contain any cycles

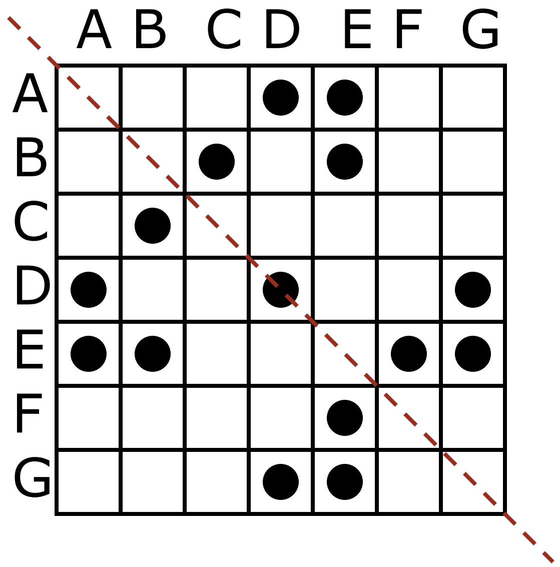
Graph Representation

- Adjacency-matrix Representation
- Adjacency Lists Representation

Adjacency-matrix representation I



- One simple way of representing a graph is the adjacency matrix
- A 2-D array has a mark at [i][j] if there is an edge from node i to node j

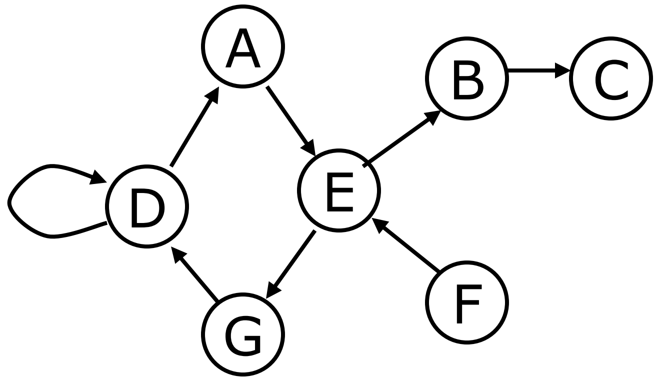


	A	B	C	D	E	F	G
A				●	●		
B			●		●		
C		●					
D	●			●			●
E	●	●				●	●
F					●		
G				●	●		

- The adjacency matrix is symmetric about the main diagonal

- This representation is only suitable for small graphs! (Why?)

Adjacency-matrix representation II



	A	B	C	D	E	F	G
A					●		
B			●				
C							
D	●			●			
E		●					●
F					●		
G				●			

- An adjacency matrix can equally well be used for digraphs (directed graphs)
- A 2-D array has a mark at [i][j] if there is an edge from node i to node j

- Again, this is only suitable for *small* graphs!

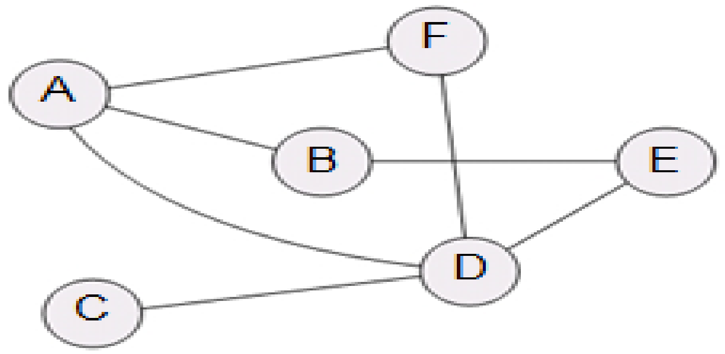
Adjacency Lists Representation

- A graph of n nodes is represented by a one-dimensional array L of linked lists, where
  - L[i] is the linked list containing all the nodes adjacent to node i
  - The nodes in the list L[i] are in no particular order

Adjacency Lists Representation

Graphs and Digraphs — Examples

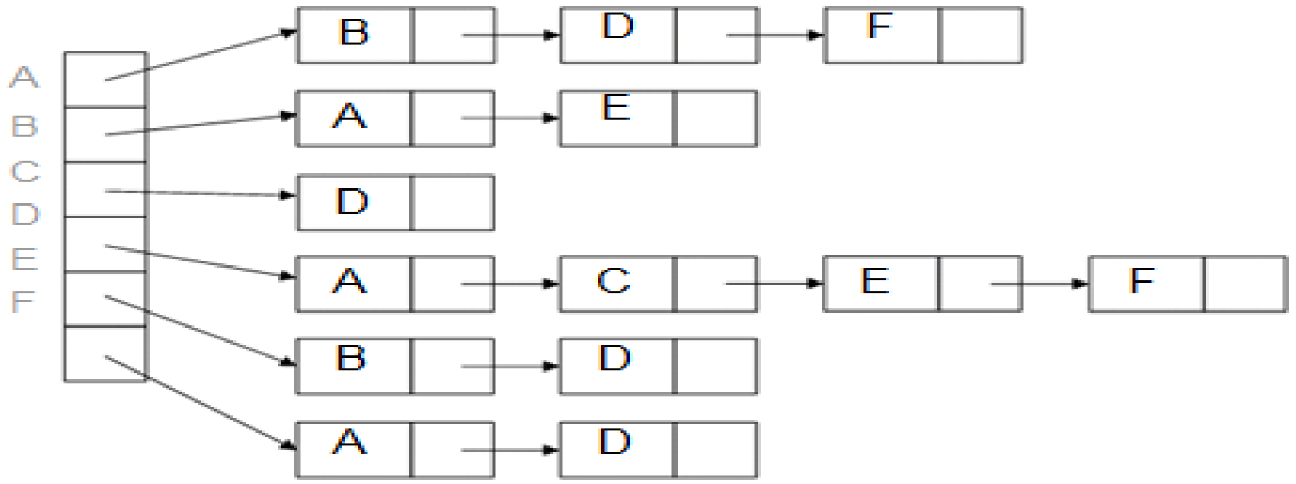
An (undirected) graph $G = (V,E)$



adjacency matrix for G

$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 0 & 1 & 0 & 1 \\ B & 1 & 0 & 0 & 0 & 1 & 0 \\ C & 0 & 0 & 0 & 1 & 0 & 0 \\ D & 1 & 0 & 1 & 0 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 & 0 \\ F & 1 & 0 & 0 & 1 & 0 & 0 \end{array}$$

adjacency list for G



Pros and Cons of Adjacency Matrix

- Pros:
  - Simple to implement
  - Easy and fast to tell if a pair (i,j) is an edge: simply check if A[i][j] is 1 or 0
- Cons:
  - No matter how few edges the graph has, the matrix takes $O(n^2)$ in memory

Pros and Cons of Adjacency Lists

Pros:

- Saves on space (memory): the representation takes as many memory words as there are nodes and edge.

Cons:

- It can take up to O(n) time to determine if a pair of nodes (i,j) is an edge: one would have to search the linked list L[i], which takes time proportional to the length of L[i].

Note:

- Adjacency list is better for sparse graphs, adjacency matrix for dense graphs

Graphs

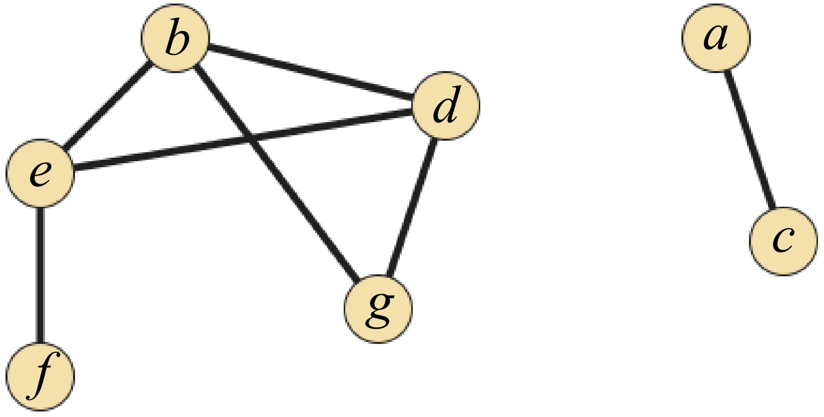
- A ***graph*** is a pair $G = (V, E)$, where
  - V is a set of vertices
  - E is a set of pairs (v, w) with v, w ∈ V
- We call the elements of V the ***vertices*** of G;
- the elements of E are the ***edges*** of G.

Vertices a and c are ***adjacent***;
vertices f and g are not.
Vertex b and edge (b, g) are ***incident***;
vertex e and edge (d, g) are not.
edge (b, d) and edge(b,e) are ***incident***
because they share a common vertex

The ***neighborhoo***d of vertex b is the
set N(b) = {d, e, g}.
The ***degree*** deg(b) of vertex b is 3.

Example:

$V = \{a, b, c, d, e, f, g\}$
$E = \{(a, c), (b, d), (b, e), (b, g),$
$(d, e), (d, g), (e, f)\}$

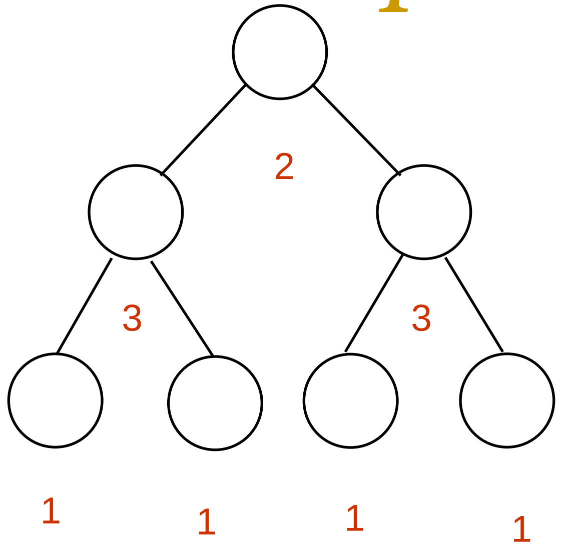


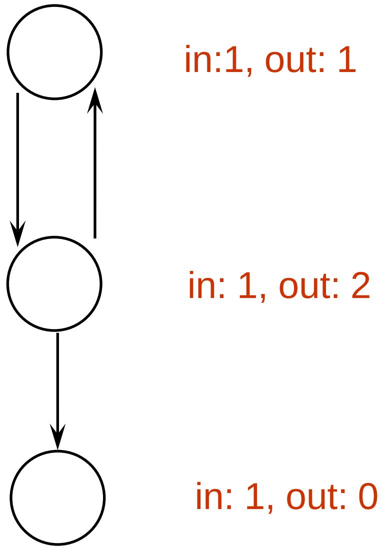
Terminology: Degree of a Vertex

- The degree of a vertex is the number of edges incident related to that vertex
- For directed graph,
- The in-degree of a vertex is the number of edges incident to it.
- The out-degree of a vertex is the number of edges incident from it.
- if ***d**$_i$* is the degree of a vertex ***i*** in a graph ***G*** with ***n*** vertices and ***e*** edges, the number of edges

$$e = (\sum_{i=0}^{n-1} d_i)/2$$

Examples





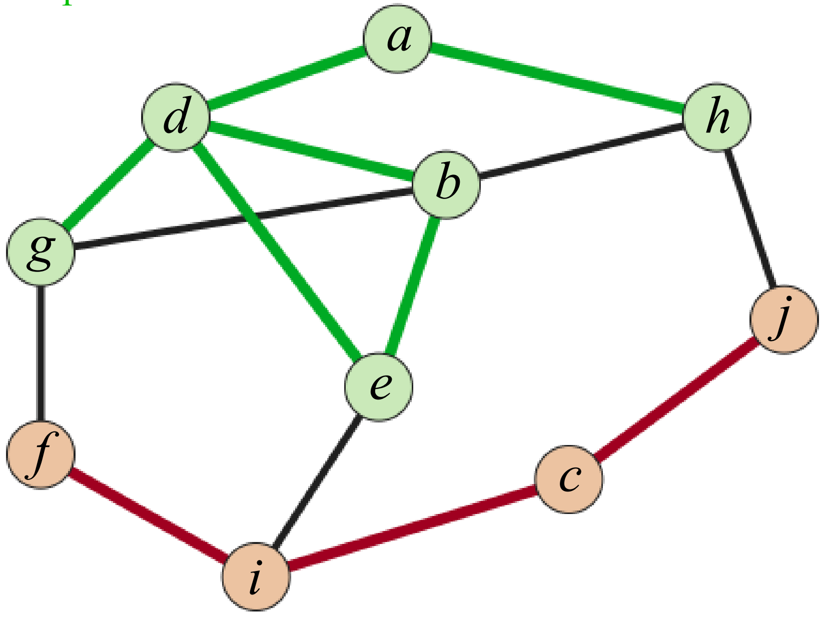
$$e = (\sum_{0}^{n-1} d_i) / 2$$

Paths and Cycles

A ***path*** is a sequence of vertices $P = (v_0, v_1, \ldots, v_k)$ such that, for $1 \le i \le k$, edge $(v_{i-1}, v_i) \in E$.

Path P is ***simple*** if no vertex appears more than once in P.

$P_1 = (g, d, e, b, d, a, h)$ is not simple.



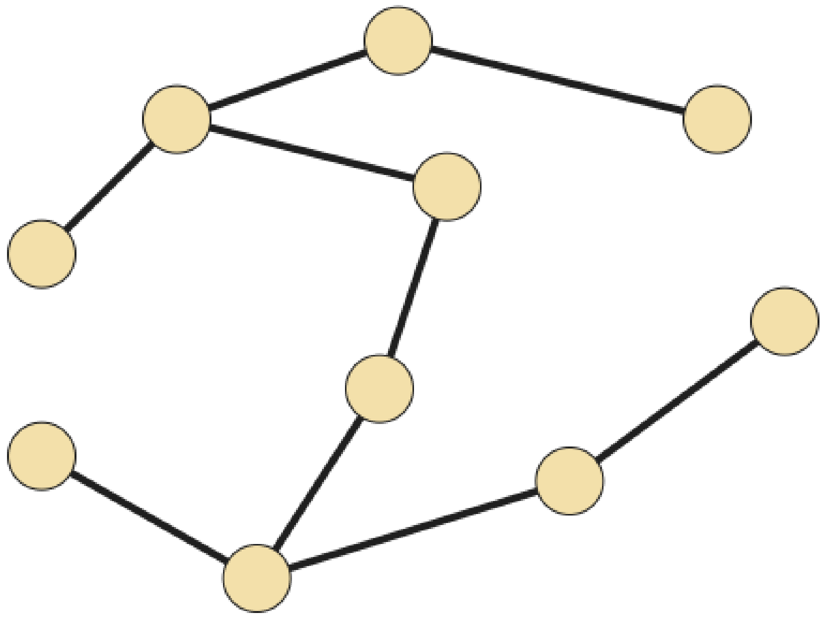
$P_2 = (f, i, c, j)$ is simple.

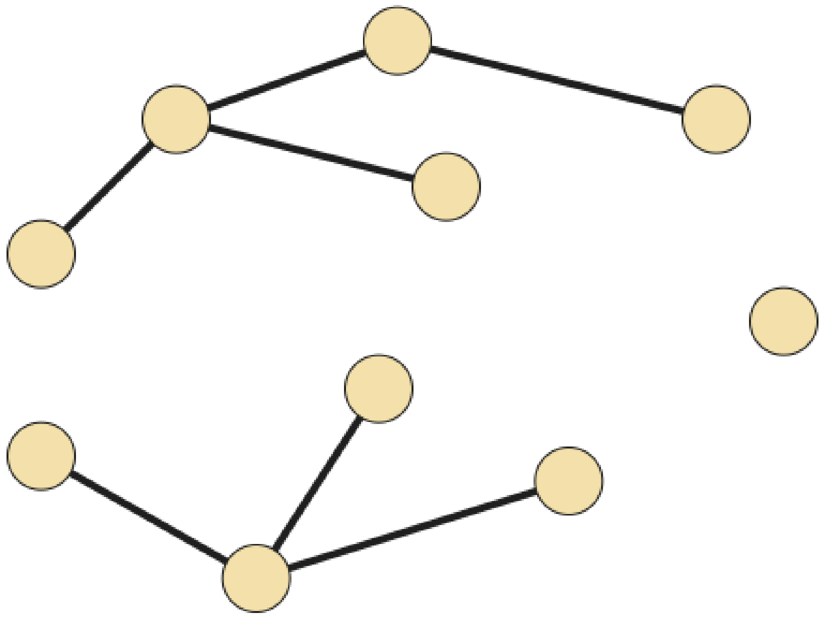
Trees and Forests

A ***tree*** is a graph that is connected and contains no cycles.

A ***forest*** is a graph that contains no cycles.

The connected components of a forest are trees.





Graph Traversal

- **Graph traversal** is the process of writing out all the nodes of a simple, connected graph *G* in some organized way.
- Algorithms, that apply for generalize traversal of any graph. They are:
- Breadth First Search
- Depth First Search

Graph Search

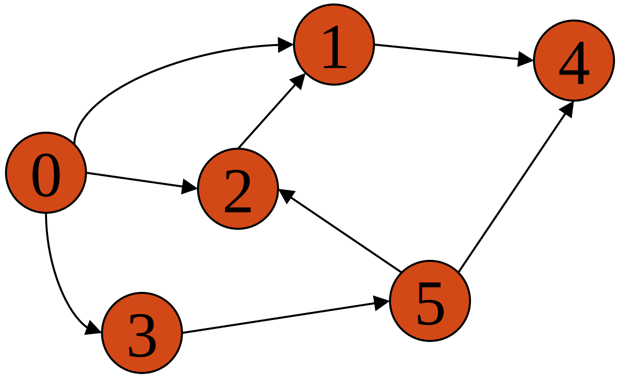
- Choice of container
  - If a stack is used as the container for adjacent vertices, we get *depth first search*.
  - If a **Queue** is used as the container adjacent vertices, we get ***breadth first search***.

Graph Search

- The state of a vertex, u, is stored in a color variable as follows:
- 1. color[u] = White - for the "undiscovered" state,
  2. color [u] = Gray - for the "discovered but not fully explored" state, and
  3. color [u] = Black - for the "fully explored" state.

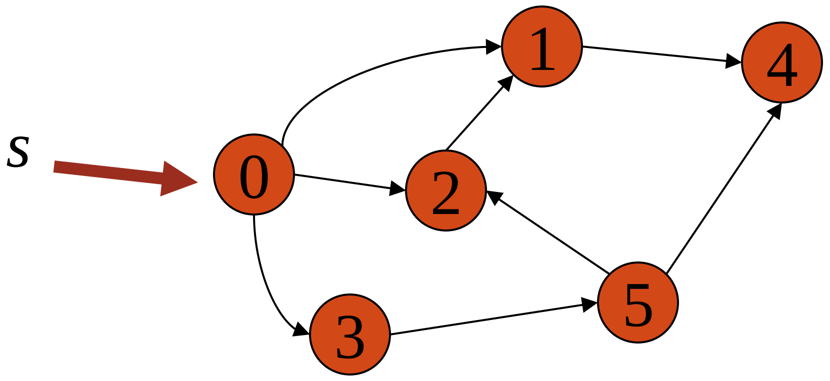
Breadth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Enqueue$(Q,v)$
            }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



Breadth First Algorithm

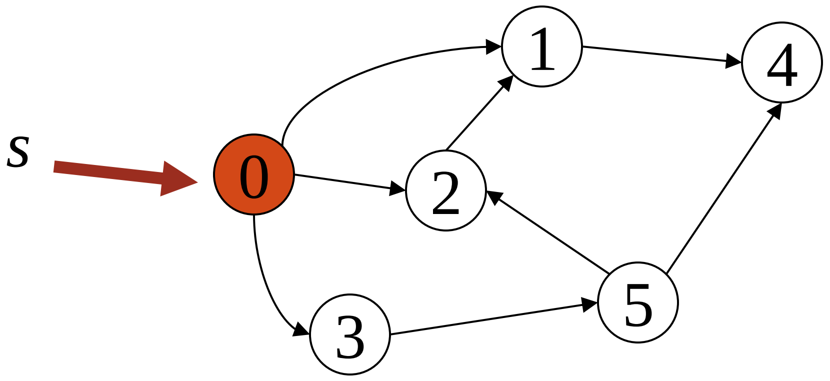
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Enqueue$(Q,v)$
            }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



$Q = \varnothing$

Breadth First Algorithm

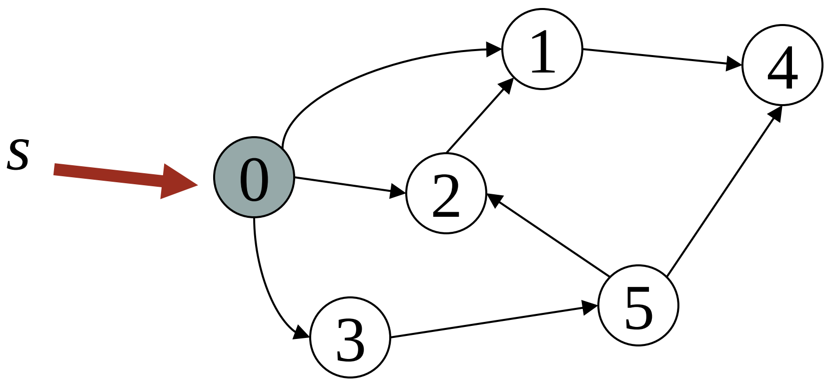
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Enqueue$(Q,v)$
            }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



$Q = \varnothing$

Breadth First Algorithm

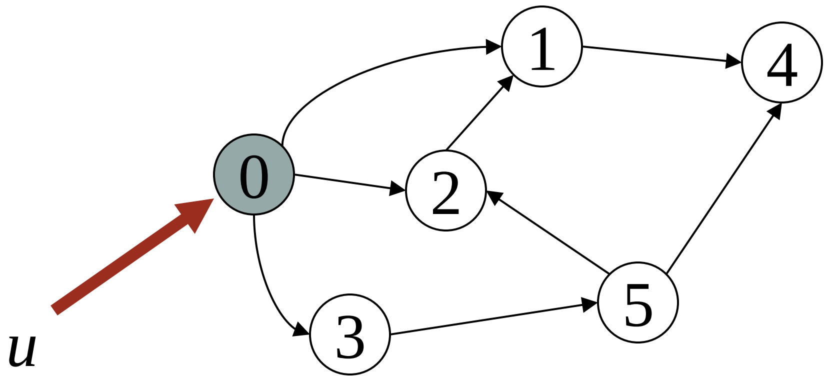
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



$Q = \boxed{0}$

Breadth First Algorithm

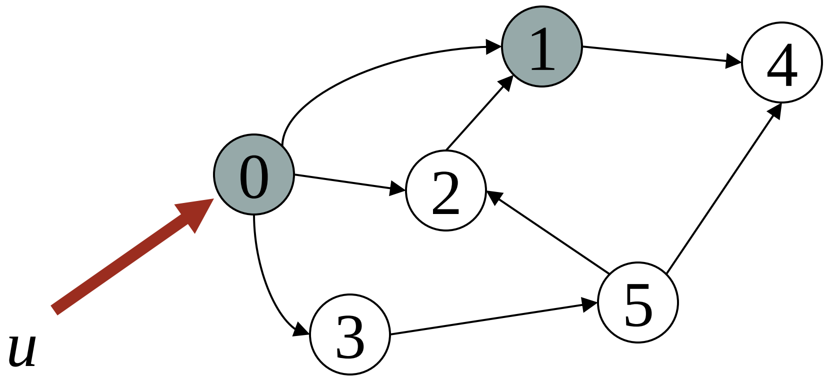
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
    {       if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Enqueue$(Q,v)$
            } }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



$Q = \boxed{0}$

Breadth First Algorithm

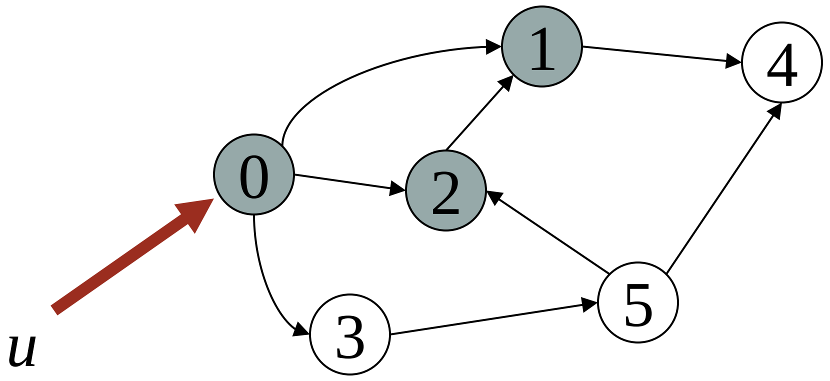
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```

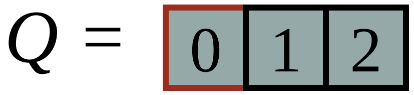


$Q = \boxed{0}\boxed{1}$

Breadth First Algorithm

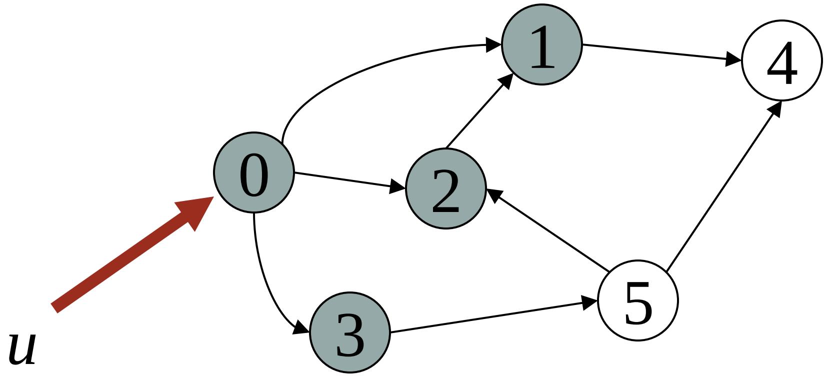
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q]$;
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black$;
}
```

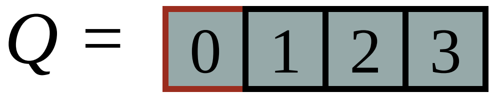




Breadth First Algorithm

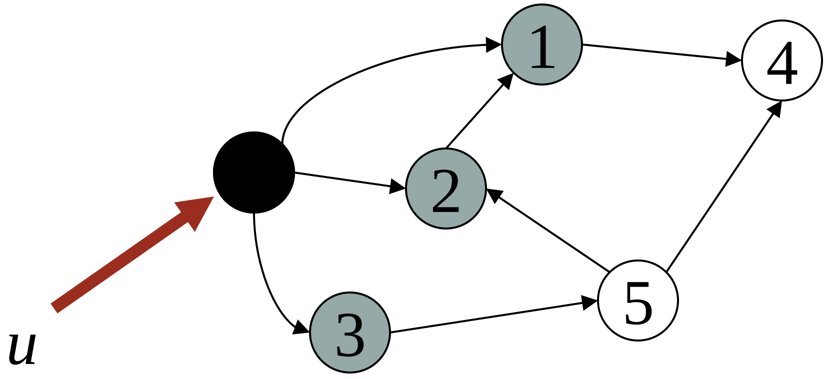
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```





Breadth First Algorithm

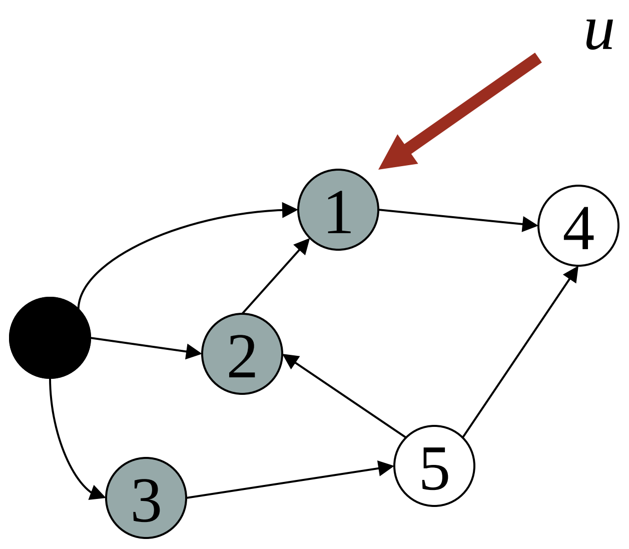
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



$Q = \boxed{1}\boxed{2}\boxed{3}$

Breadth First Algorithm

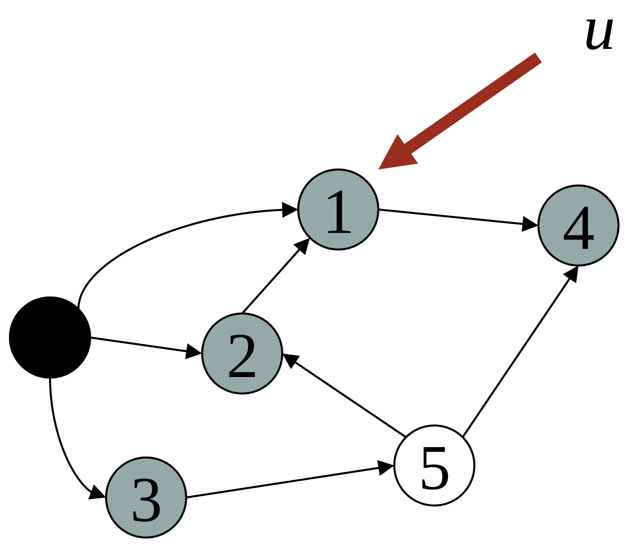
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```

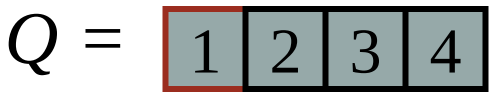


$Q = \boxed{1}\boxed{2}\boxed{3}$

Breadth First Algorithm

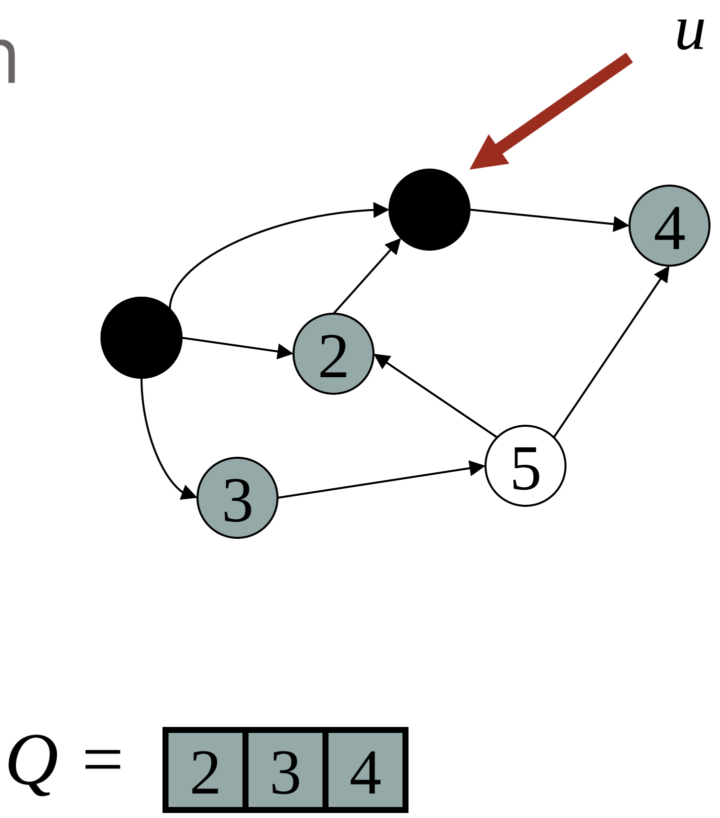
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Enqueue$(Q,v)$
            }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```





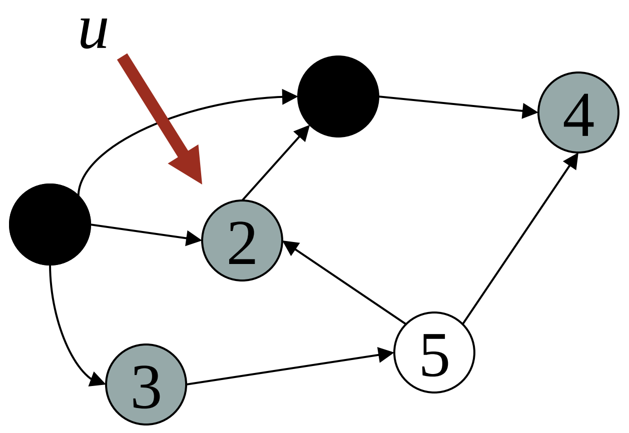
Breadth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q]$;
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q, v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black$;
}
```



Breadth First Algorithm

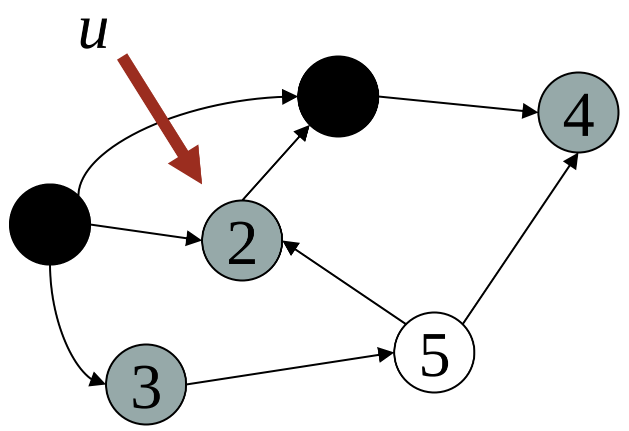
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```

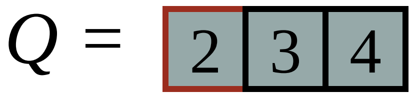


$Q =$ | 2 | 3 | 4 |

Breadth First Algorithm

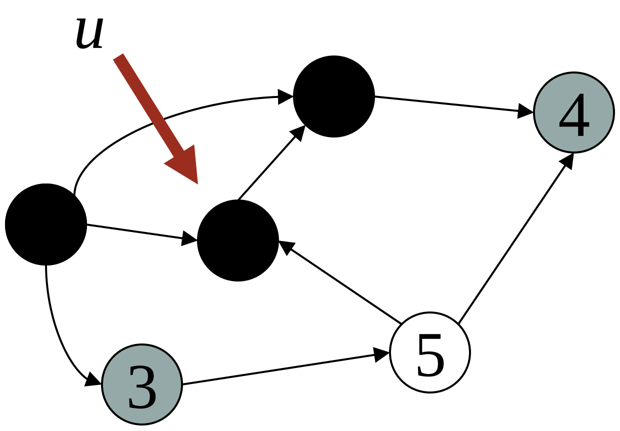
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```

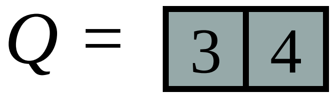




Breadth First Algorithm

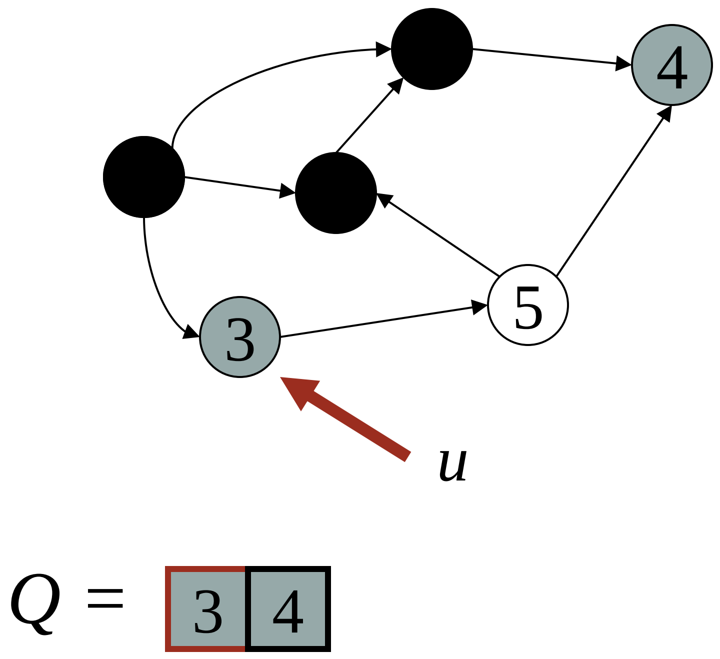
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```





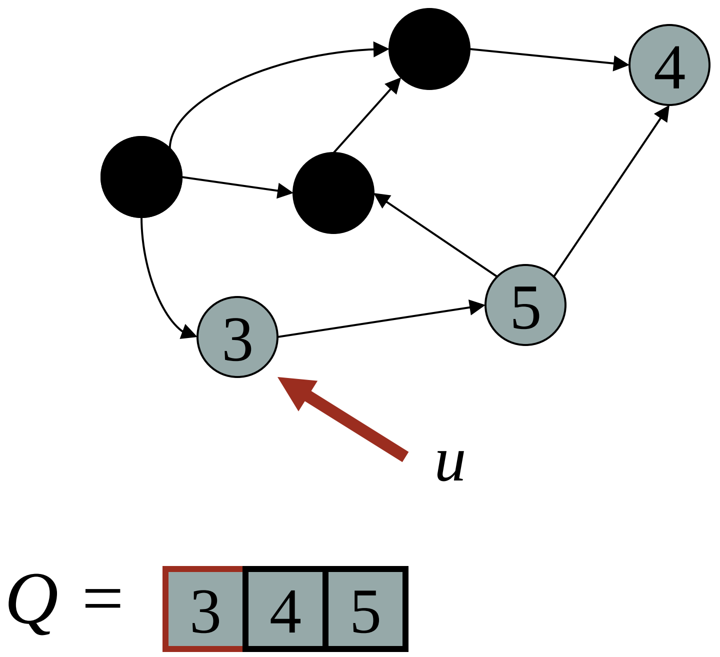
Breadth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



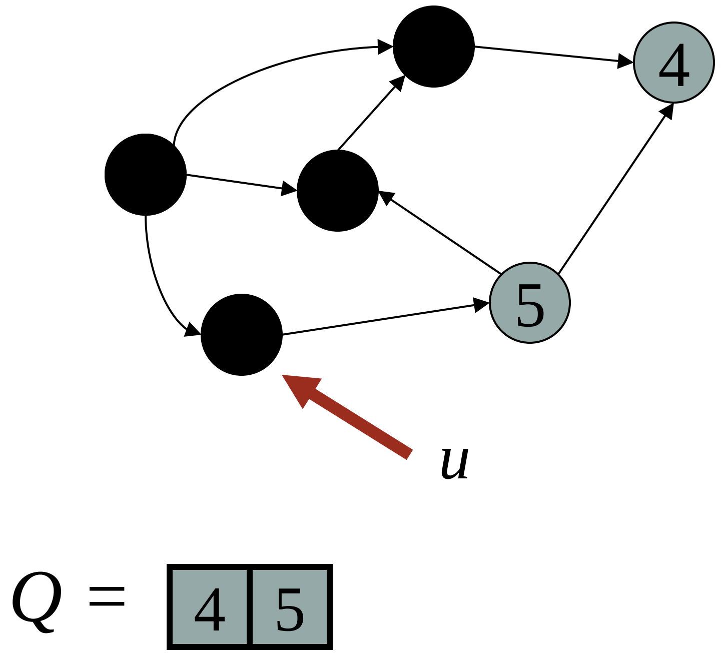
Breadth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Enqueue$(Q,v)$
            }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



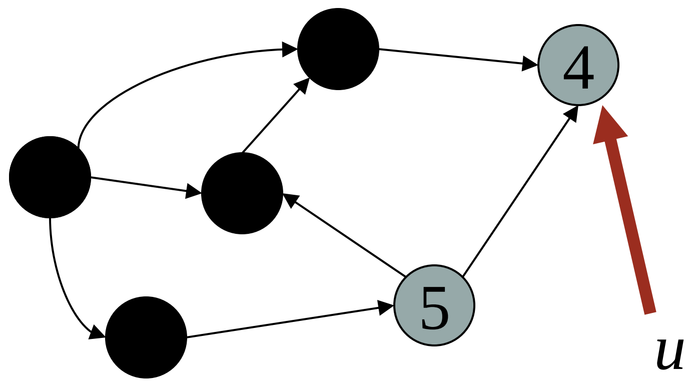
Breadth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



Breadth First Algorithm

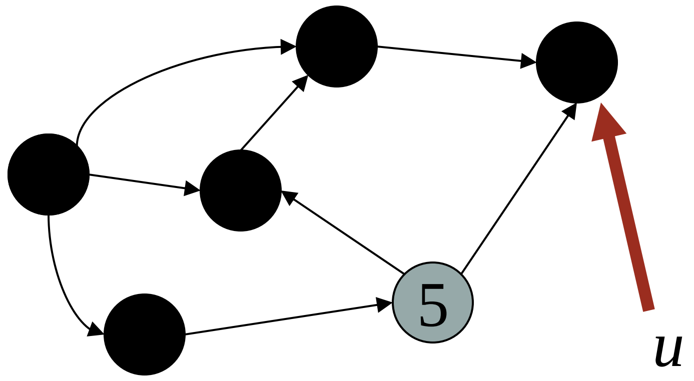
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



$Q = \boxed{4}\boxed{5}$

Breadth First Algorithm

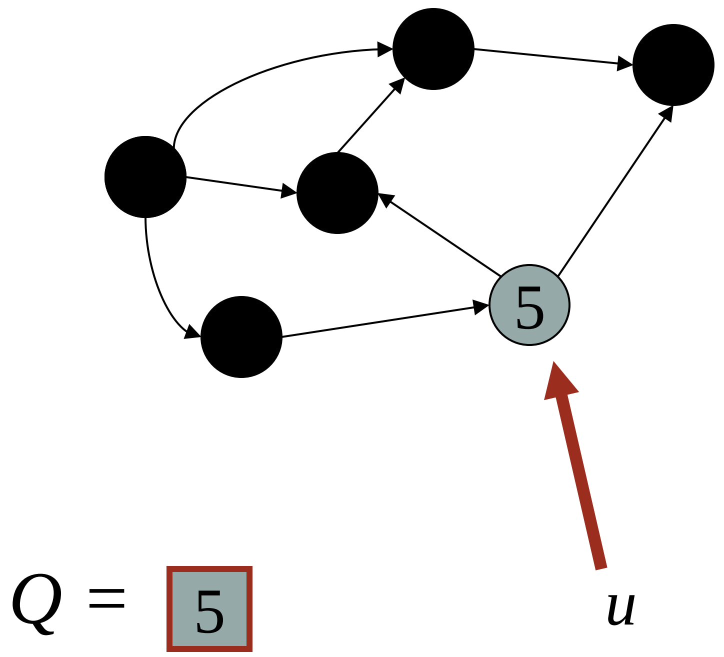
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \emptyset$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



$Q = \boxed{5}$

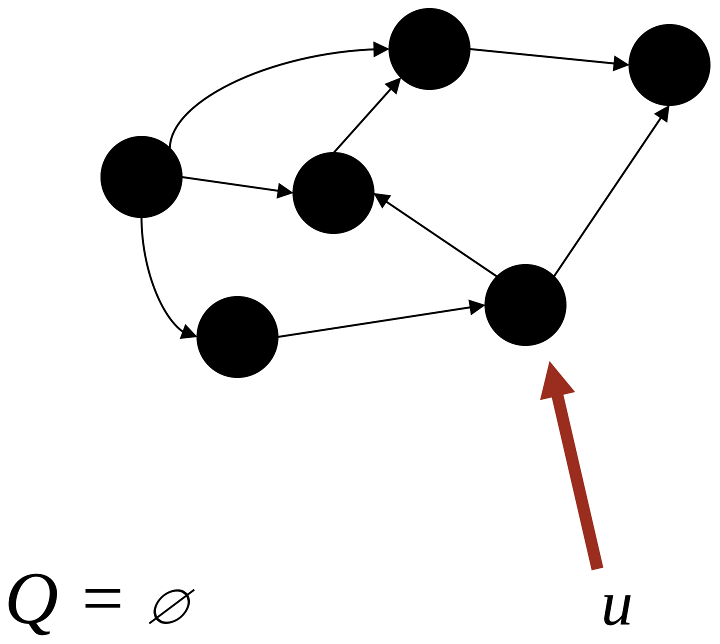
Breadth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



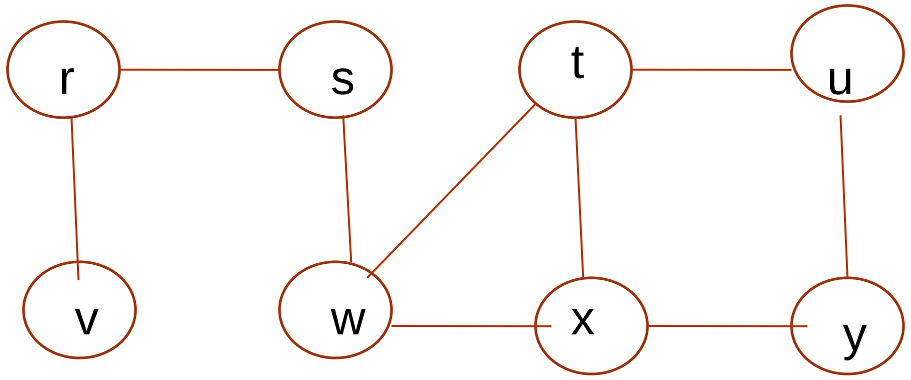
Breadth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a queue Q
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$Q \leftarrow \{s\}$
While $Q \neq \varnothing$
{
    $u \leftarrow head[Q];$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Enqueue$(Q,v)$
        }
    Dequeue$(Q)$
    $color[u] \leftarrow black;$
}
```



- Ignore Vertex:
  - If an edge goes to completely explored vertex (black)
  - If an edge goes to discovered but not completely explored (Grey)

Task
Example of Undirected Graph



- Root is s

Analysis Of Breadth First Algorithm

- The while-loop in breadth-first search is executed at most |V| times. The reason is that every vertex enqueued at most once. So, we have O(V).
- The for-loop inside the while-loop is executed at most |E| times if G is a directed graph or 2|E| times if G is undirected. In Directed graph we examine (*u*, *v*) only when *u* is dequeued. Therefore, every edge examined at most once if directed, at most twice if undirected. So, we have O(E).
- Therefore, the total running time for breadth-first search traversal is O(V + E).

- Ignore Vertex:
  - If an edge goes to completely explored vertex (black)
  - If an edge goes to discovered but not completely explored (Grey)

Graph Search

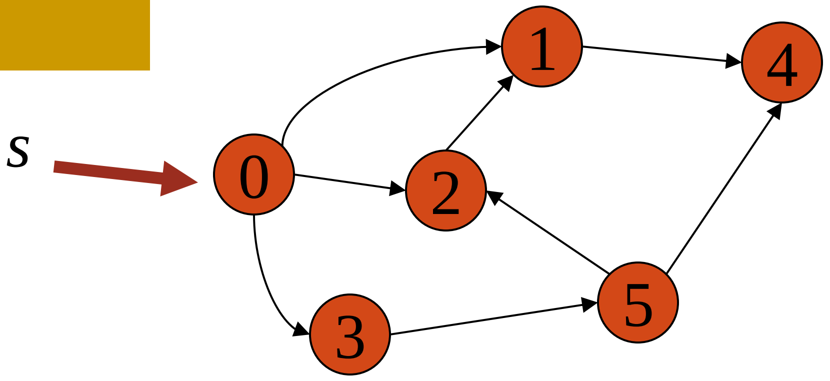
- Choice of container
  - If a **stack** is used as the container for adjacent vertices, we get ***depth first search.***
  - If a Queue is used as the container adjacent vertices, we get *breadth first search.*

Depth First Algorithm

Given graph $G=(V,E)$ and source vertex $s \in V$

Create a stack S

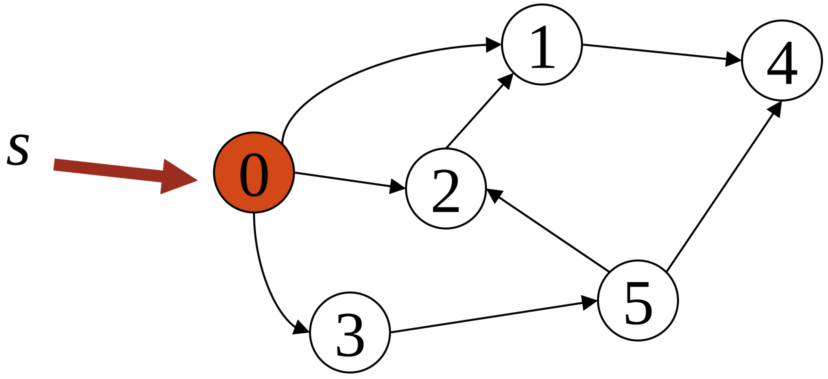
```
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u$ = Pop$(S)$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push$(S,v)$
        }
    $color[u] \leftarrow black;$
}
```



$S = \emptyset$

Depth First Algorithm

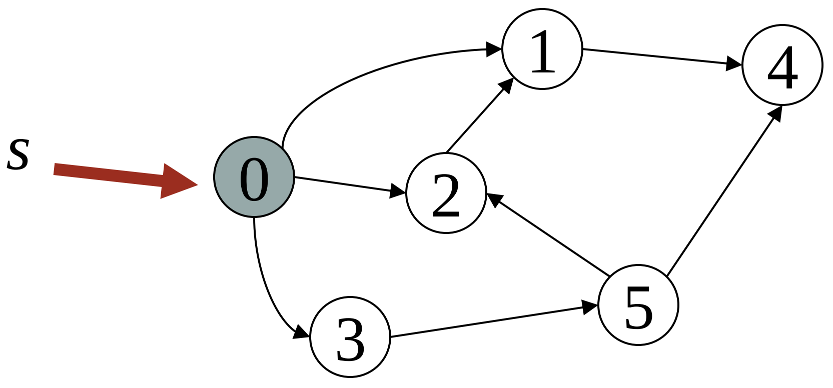
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
        $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Push($S,v$)
            }
    $color[u] \leftarrow black;$
}
```



$S = \emptyset$

Depth First Algorithm

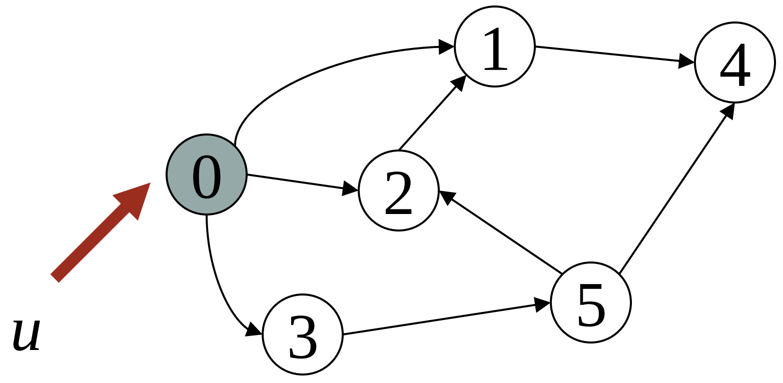
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u = \text{Pop}(S)$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            $\text{Push}(S, v)$
        }
    $color[u] \leftarrow black;$
}
```



$S = \boxed{0}$

Depth First Algorithm

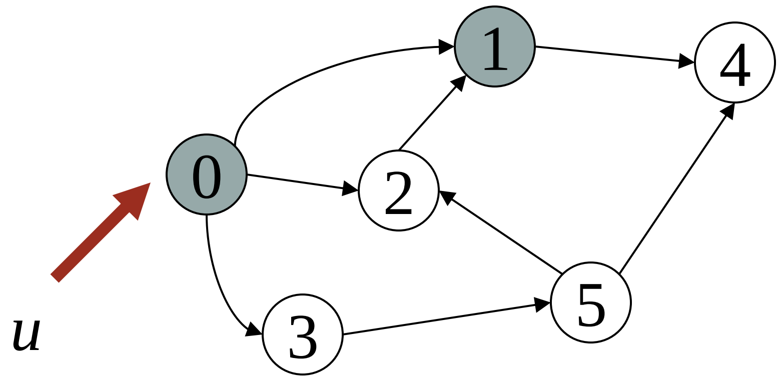
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S,v$)
        }
    $color[u] \leftarrow black;$
}
```



$S = \varnothing$

Depth First Algorithm

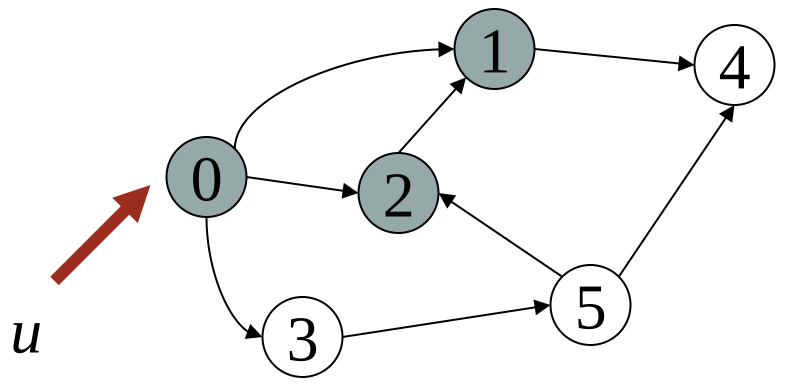
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Push($S, v$)
            }
    $color[u] \leftarrow black;$
}
```



$S = \boxed{1}$

Depth First Algorithm

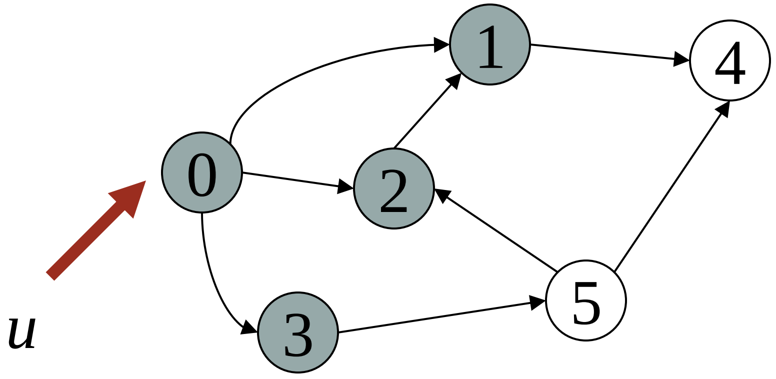
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S,v$)
        }
    $color[u] \leftarrow black;$
}
```



$S = \boxed{2}\boxed{1}$

Depth First Algorithm

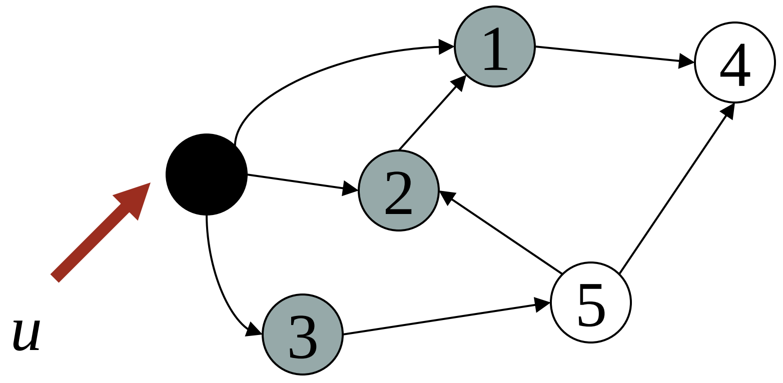
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S,v$)
        }
    $color[u] \leftarrow black;$
}
```

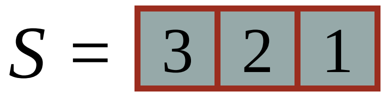


$S = \boxed{3}\boxed{2}\boxed{1}$

Depth First Algorithm

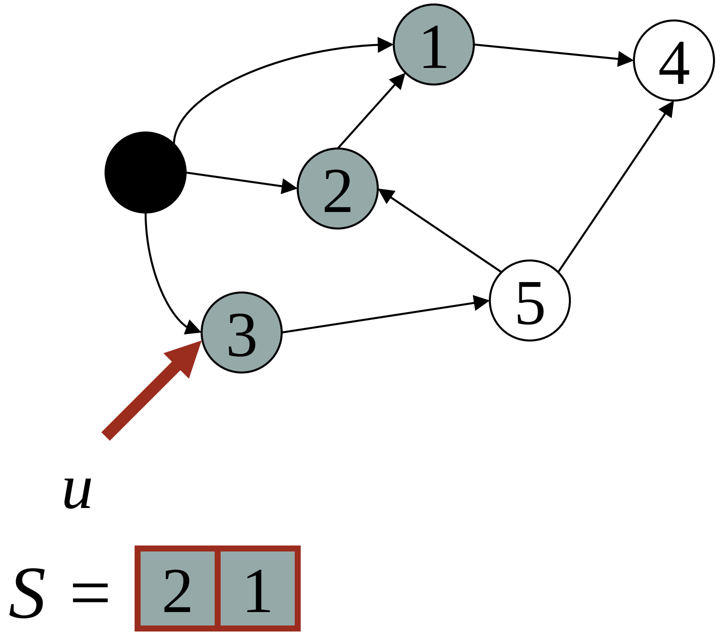
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S,v$)
        }
    $color[u] \leftarrow black;$
}
```





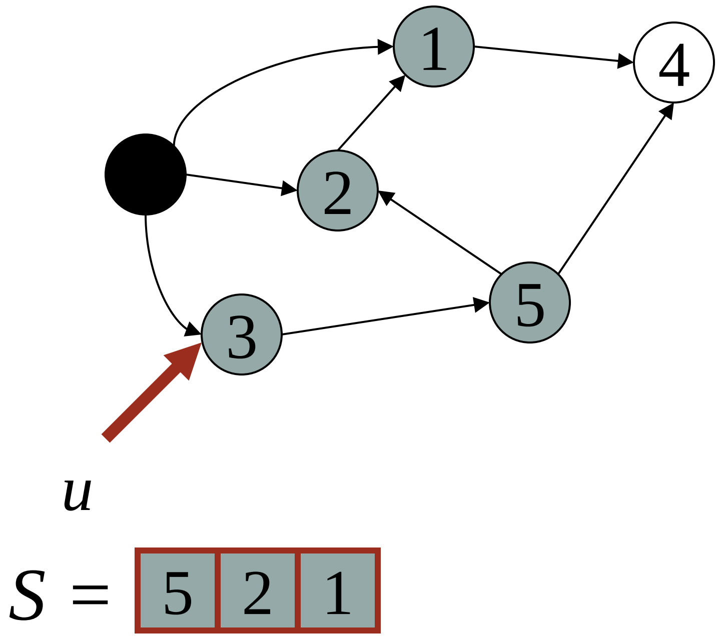
Depth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u = \text{Pop}(S)$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            $\text{Push}(S,v)$
        }
    $color[u] \leftarrow black;$
}
```



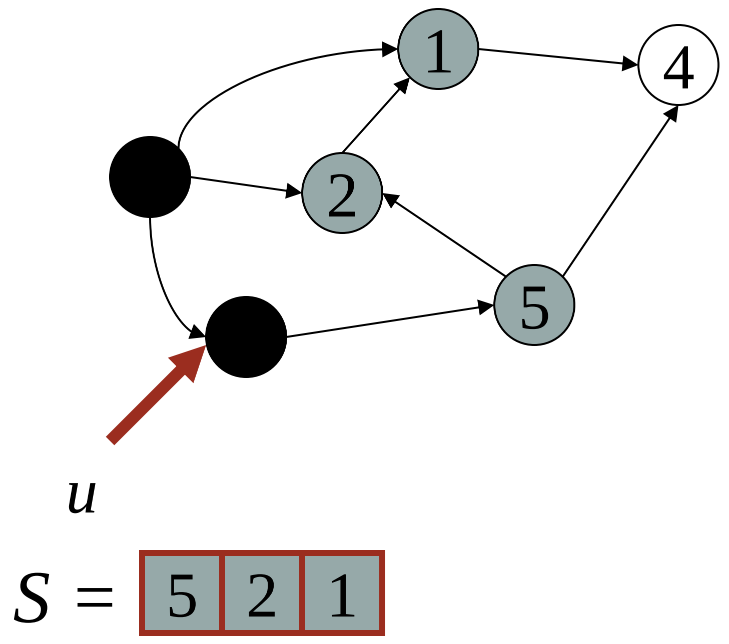
Depth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u = Pop(S)$
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                $Push(S,v)$
            }
    $color[u] \leftarrow black;$
}
```



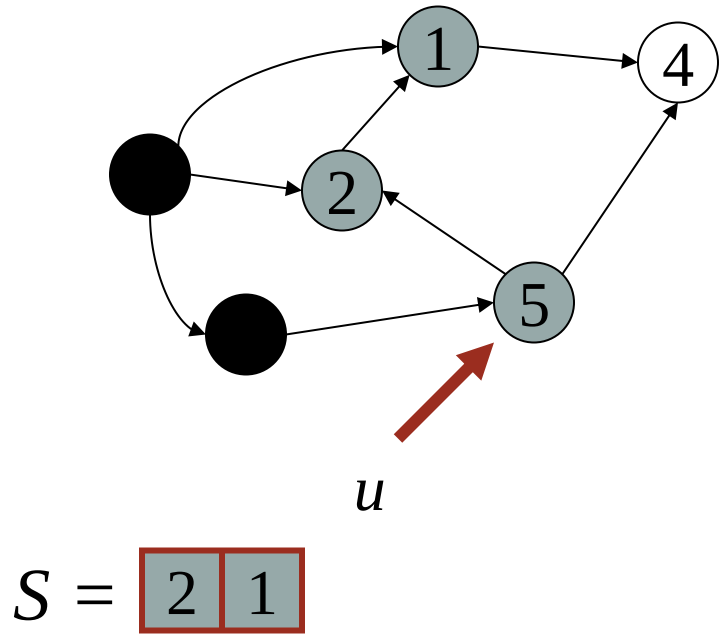
Depth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S, v$)
        }
    $color[u] \leftarrow black;$
}
```



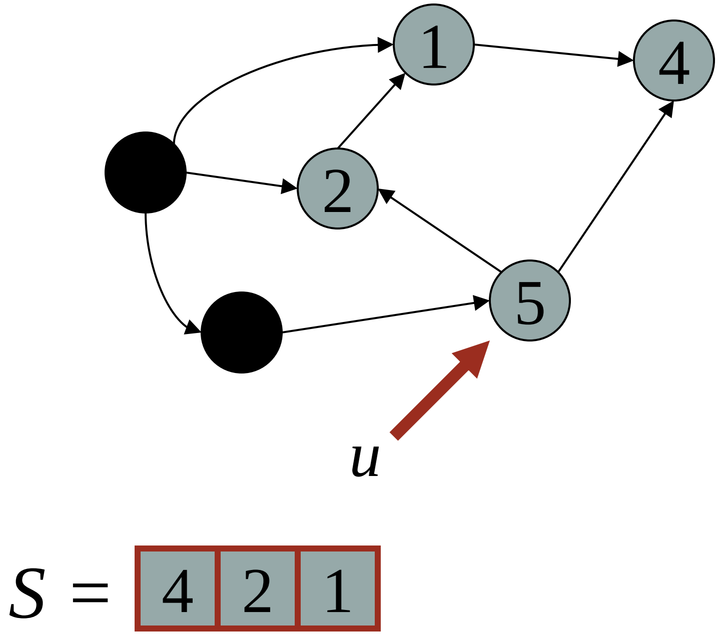
Depth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u = \text{Pop}(S)$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push$(S, v)$
        }
    $color[u] \leftarrow black;$
}
```



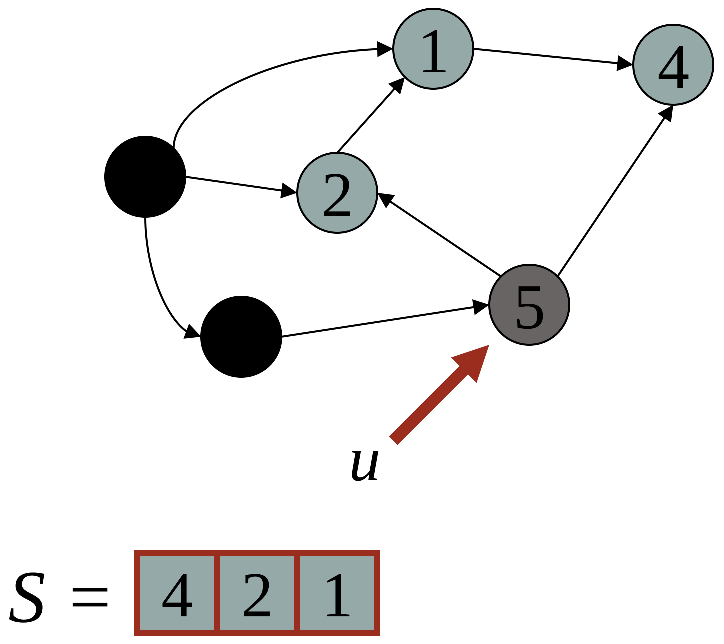
Depth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
            if $color[v] = white$
            {
                $color[v] \leftarrow gray$
                Push($S,v$)
            }
    $color[u] \leftarrow black;$
}
```



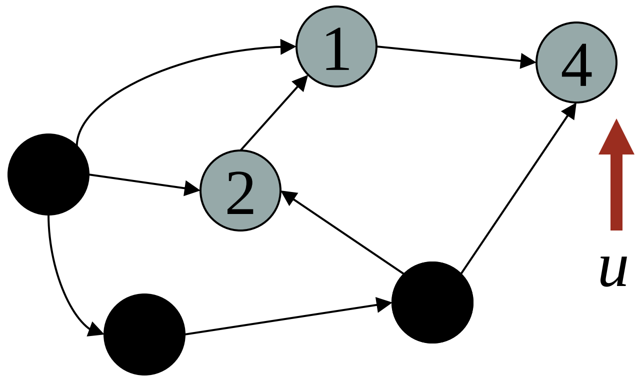
Depth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S, v$)
        }
    $color[u] \leftarrow black;$
}
```



Depth First Algorithm

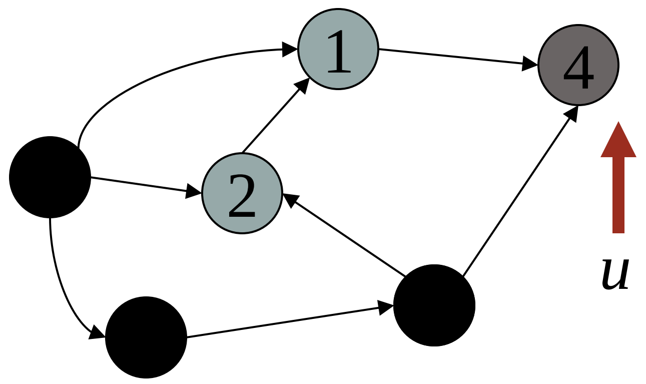
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u = \text{Pop}(S)$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            $\text{Push}(S, v)$
        }
    $color[u] \leftarrow black;$
}
```

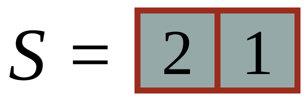


$S = \boxed{2}\boxed{1}$

Depth First Algorithm

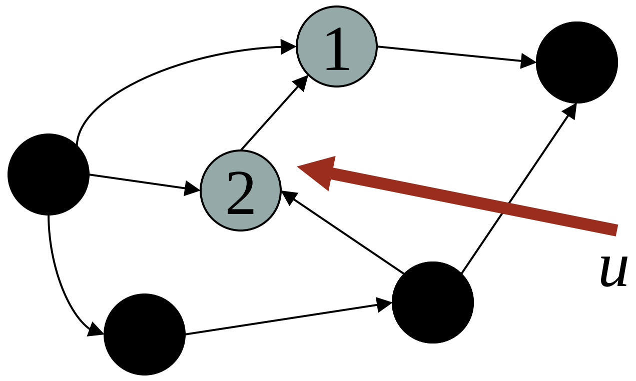
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S, v$)
        }
    $color[u] \leftarrow black;$
}
```



S =


Depth First Algorithm

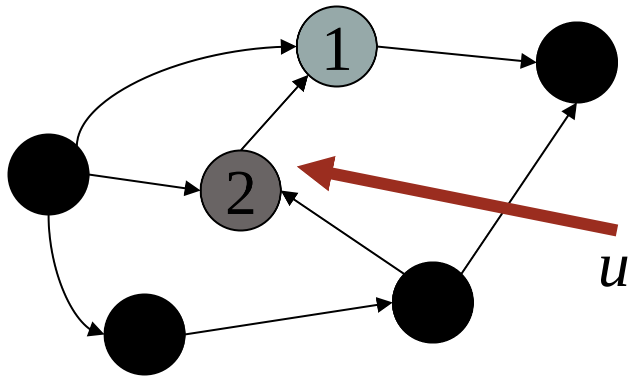
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S, v$)
        }
    $color[u] \leftarrow black;$
}
```



$S = \boxed{1}$

Depth First Algorithm

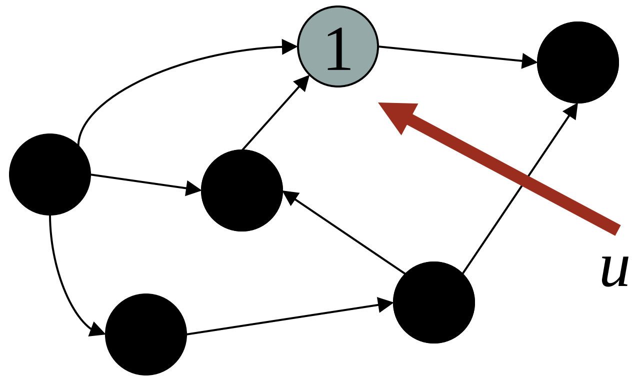
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S, v$)
        }
    $color[u] \leftarrow black;$
}
```



$S = \boxed{1}$

Depth First Algorithm

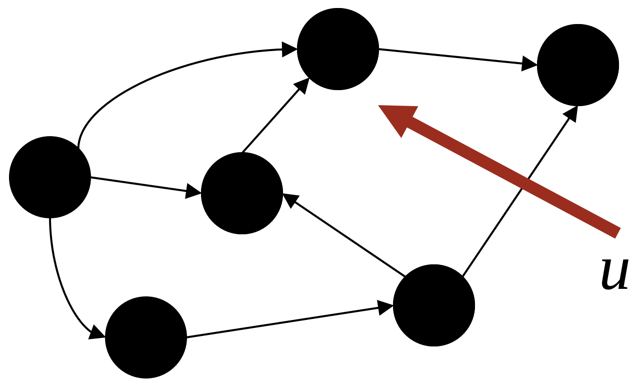
```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \emptyset$
{
    $u = \text{Pop}(S)$
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push$(S,v)$
        }
    $color[u] \leftarrow black;$
}
```



$S = \emptyset$

Depth First Algorithm

```
Given graph $G=(V,E)$ and source vertex $s \in V$
Create a stack $S$
For each vertex $u \in V - \{s\}$
    $color[u] \leftarrow white$
$color[s] \leftarrow gray$
$S \leftarrow \{s\}$
While $S \neq \varnothing$
{
    $u$ = Pop($S$)
    for each $v \in Adjacent[u]$
        if $color[v] = white$
        {
            $color[v] \leftarrow gray$
            Push($S,v$)
        }
    $color[u] \leftarrow black;$
}
```



$S = \varnothing$

Analysis Of DFS and BFS

- The for-loop inside the while-loop is executed at most |E| times if G is a directed graph or 2|E| times if G is undirected. In Directed graph we examine (*u*, *v*) only when *u* is dequeued. Therefore, every edge examined at most once if directed, at most twice if undirected. So, we have O(E).
- Therefore, the total running time for breadth-first search traversal is O(V + E).
- If adjacency matrix is used as a container analysis will be $O(V^2)$

V+E ?

- v1 + (incident edges) + v2 + (incident edges) + …. + vn + (incident edges)

- (v1 + v2 + … + vn) + [(incident_edges v1) + (incident_edges v2) + … + (incident_edges vn)]

Application of DFS

- Topological ordering
- Finding cycle in directed graph
- Find strongly connected components

BASIS FOR COMPARISON	BFS	DFS
Basic	Vertex-based algorithm	Edge-based algorithm
Data structure used to store the nodes	Queue	Stack
Memory consumption	Inefficient	Efficient
Structure of the constructed tree	Wide and short	Narrow and long
Traversing fashion	Oldest unvisited vertices are explored at first.	Vertices along the edge are explored in the beginning.
Optimality	Optimal for finding the shortest distance, not in cost.	Not optimal
Application	Examines bipartite graph, connected component and shortest path present in a graph.	Examines two-edge connected graph, strongly connected graph, acyclic graph and topological order.

Key Differences Between BFS/DFS

1. BFS is vertex-based algorithm while DFS is an edge-based algorithm.
2. Queue data structure is used in BFS. On the other hand, DFS uses stack or recursion.
3. Memory space is efficiently utilized in DFS while space utilization in BFS is not effective.
4. BFS is optimal algorithm while DFS is not optimal.
5. DFS constructs narrow and long trees. As against, BFS constructs wide and short tree.

Task : Print cycles