Data Structures

Graphs Traversal

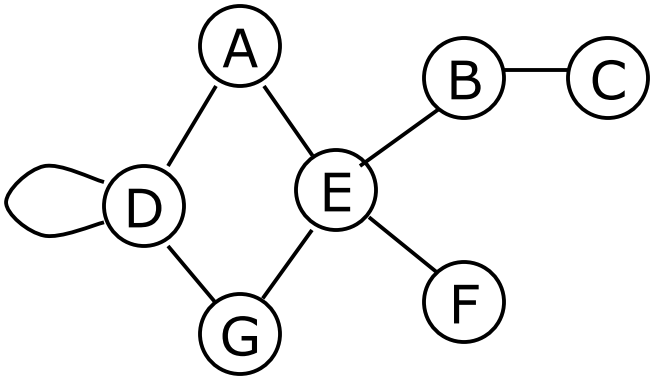
Breadth First Search (BFS)

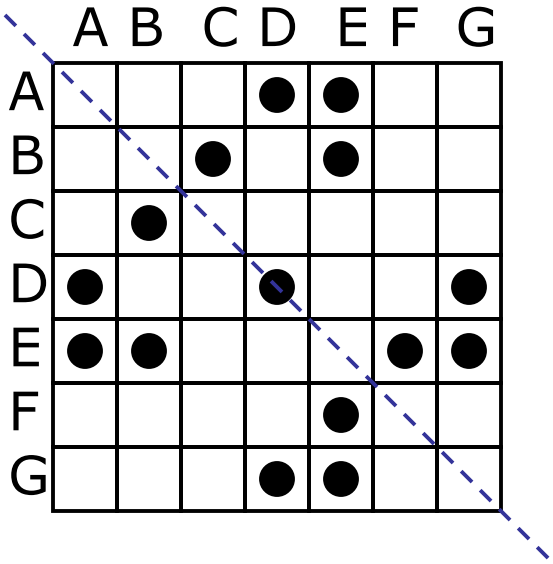
Depth First Search (DFS)

Graph Representation

- Adjacency-matrix Representation
- Adjacency Lists Representation

Adjacency-matrix representation I



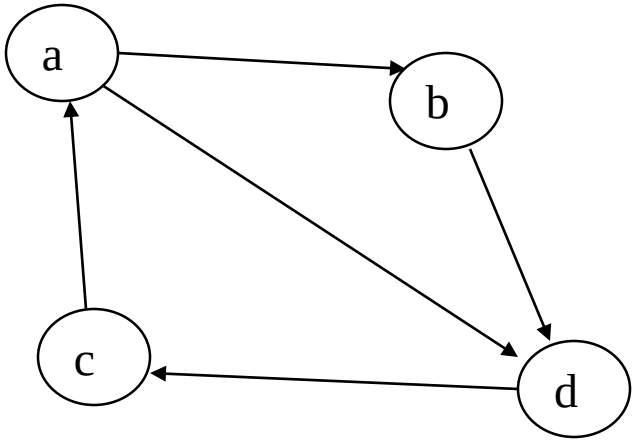


- One simple way of representing a graph is the adjacency matrix
- A 2-D array has a mark at [i][j] if there is an edge from node i to node j

- The adjacency matrix is symmetric about the main diagonal

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

Adjacency Matrix – Un weighted graph Example



	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	1	0	0	0
d	0	0	1	0

Adjacency Matrix – weighted graph Example

.vertices

[0]	"Atlanta "
[1]	"Austin "
[2]	"Chicago "
[3]	"Dallas "
[4]	"Denver "
[5]	"Houston "
[6]	"Washington"

.edges

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	0	0	0	0	0	800	600
[1]	0	0	0	200	0	160	0
[2]	0	0	0	0	1000	0	0
[3]	0	200	900	0	780	0	0
[4]	1400	0	1000	0	0	0	0
[5]	800	0	0	0	0	0	0
[6]	600	0	0	1300	0	0	0

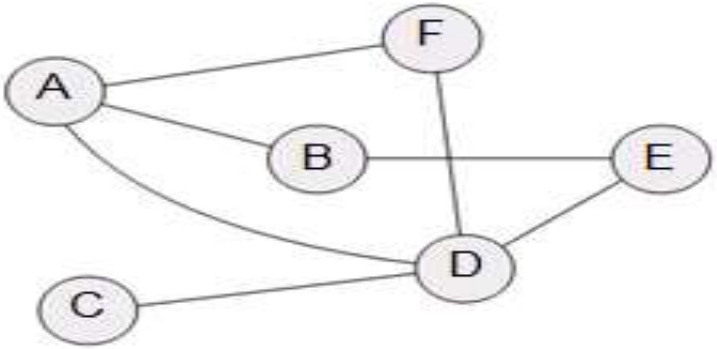
Representing Graphs: Adjacency List

- **Adjacency List**
  - Array of lists
  - Each vertex has an array entry
  - A vertex w is inserted in the list for vertex v if there is an outgoing edge from v to w

Adjacency Lists Representation

Graphs and Digraphs — Examples

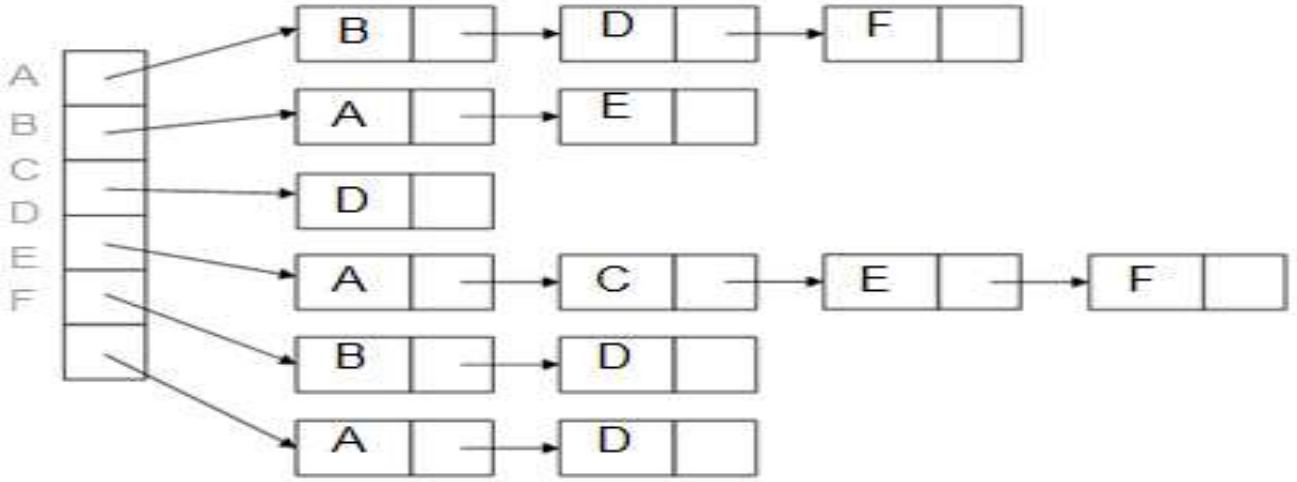
An (undirected) graph $G = (V,E)$



adjacency matrix for G

	A	B	C	D	E	F
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

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 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].

Graph Traversal

Given: a graph G = (V, E), directed or undirected

Goal: methodically explore every vertex and every edge

Two main approaches:

- Breadth-First Search
- Depth-First Search

Breadth First Search

- The algorithm discovers all vertices at distance k from s before discovering any vertices at distance k+1

Breadth First Search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Enqueue the root node
2. Dequeue a node and examine it
    1. If the element required is found in this node, quit the search and return a result.
    2. Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
4. If the queue is not empty, repeat from Step 2.

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 $color[v] = white$

{

$color[v] \leftarrow gray$

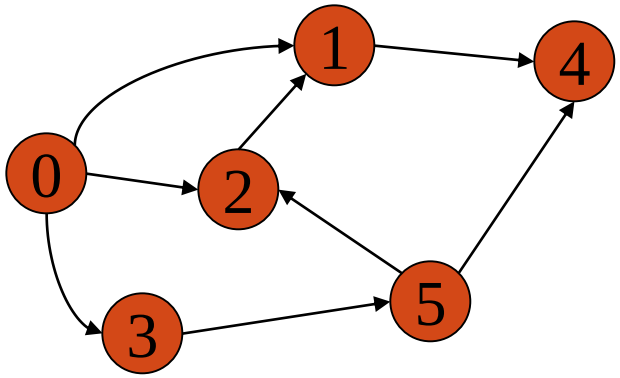
Enqueue(Q,v)

}

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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

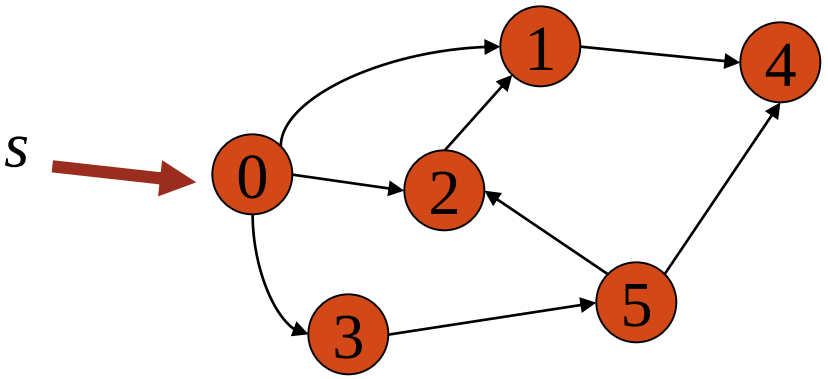
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q = \varnothing$

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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

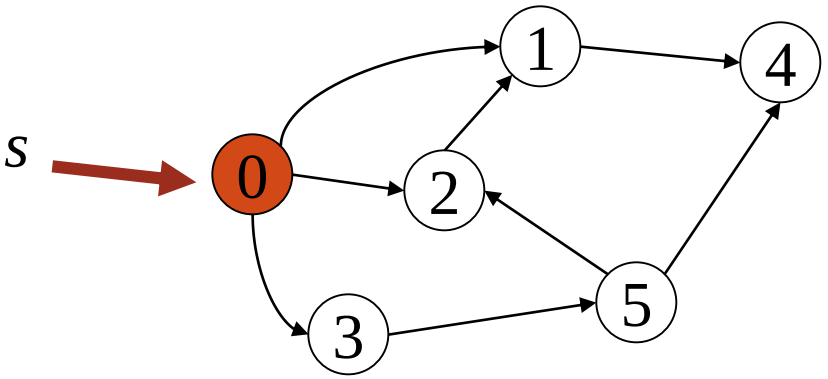
            Enqueue(Q,v)

        }

    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 $Q \neq \varnothing$

{

    $u \leftarrow head[Q];$

    for each $v \in Adjacent[u]$

        if $color[v] = white$

        {

            $color[v] \leftarrow gray$

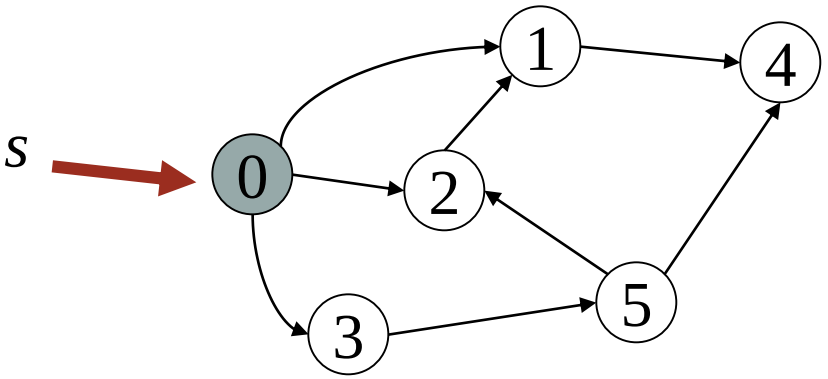
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

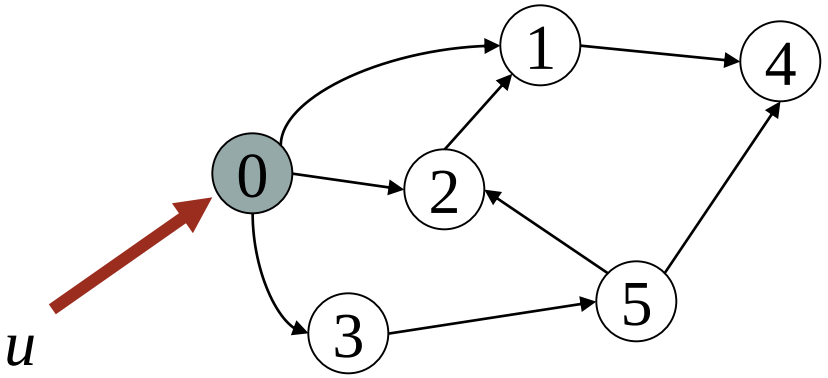
            Enqueue(Q,v)

        } }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

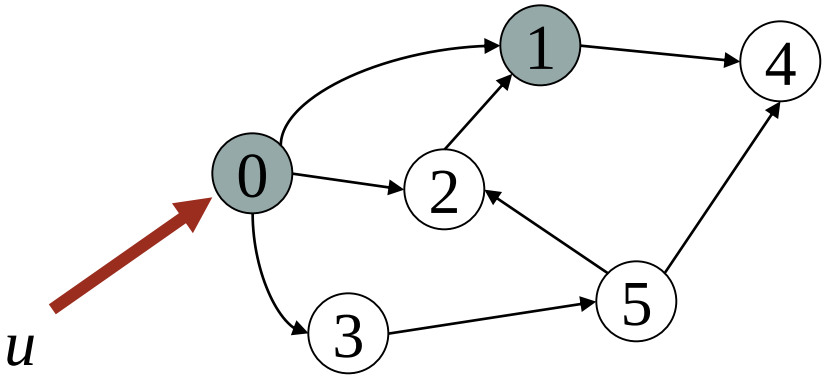
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 

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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

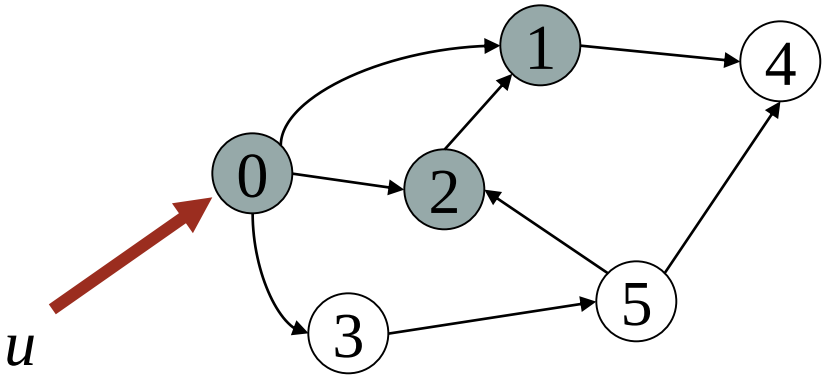
            Enqueue(Q,v)

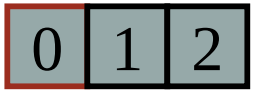
        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 

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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

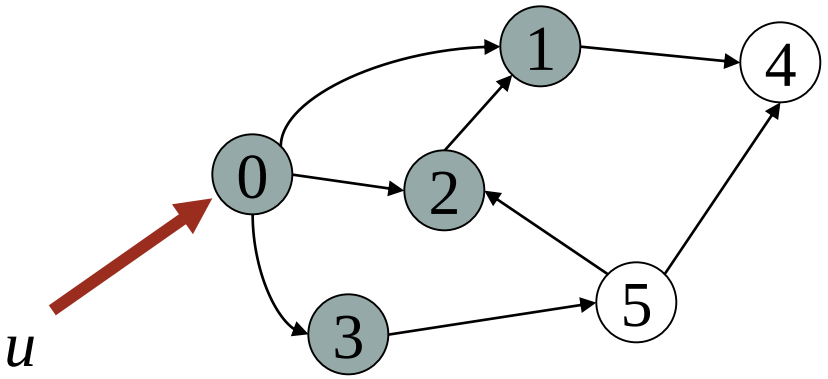
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ | 0 | 1 | 2 | 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 $Q \neq \varnothing$

{

    $u \leftarrow head[Q];$

    for each $v \in Adjacent[u]$

        if $color[v] = white$

        {

            $color[v] \leftarrow gray$

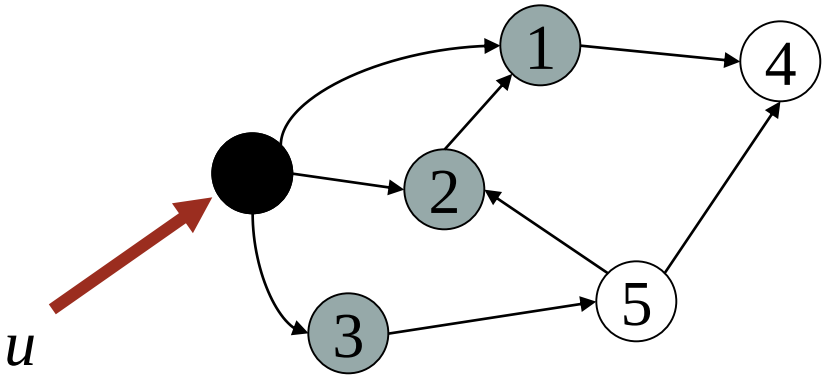
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ | 1 | 2 | 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 $Q \neq \varnothing$

{

    $u \leftarrow head[Q];$

    for each $v \in Adjacent[u]$

        if $color[v] = white$

        {

            $color[v] \leftarrow gray$

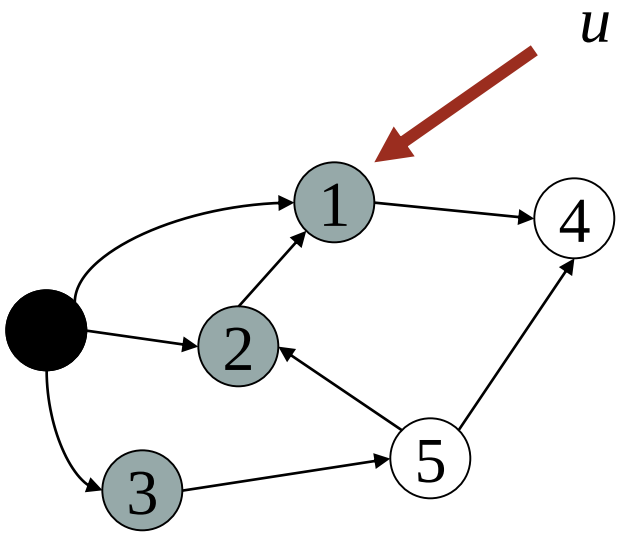
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 

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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

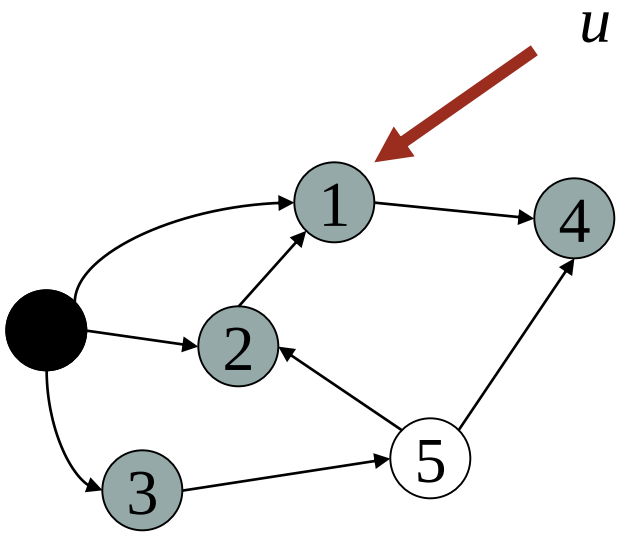
            Enqueue(Q,v)

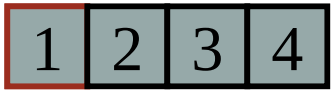
        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 

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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

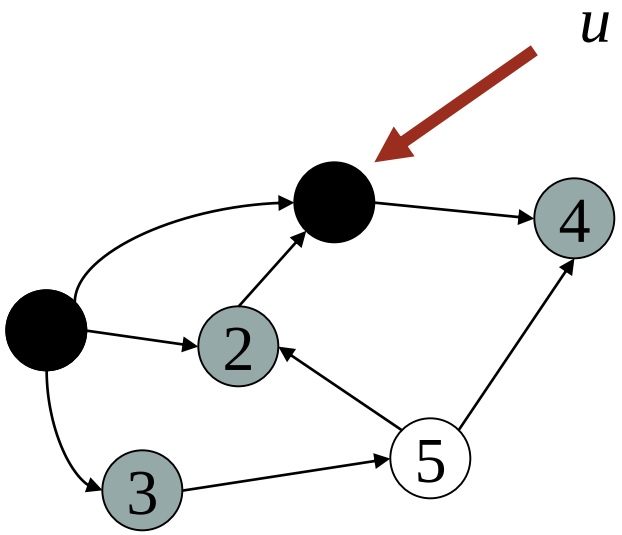
            Enqueue(Q,v)

        }

    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 $Q \neq \varnothing$

{

    $u \leftarrow head[Q];$

    for each $v \in Adjacent[u]$

        if $color[v] = white$

        {

            $color[v] \leftarrow gray$

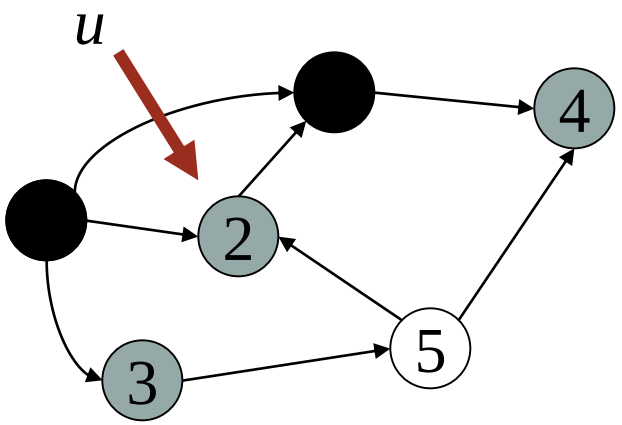
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 

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] ← white

color[s] ← gray

Q ← {s}

While $Q \neq \varnothing$

{

    u ← head[Q];

    for each v ∈ Adjacent[u]

        if color[v] = white

        {

            color[v] ← gray

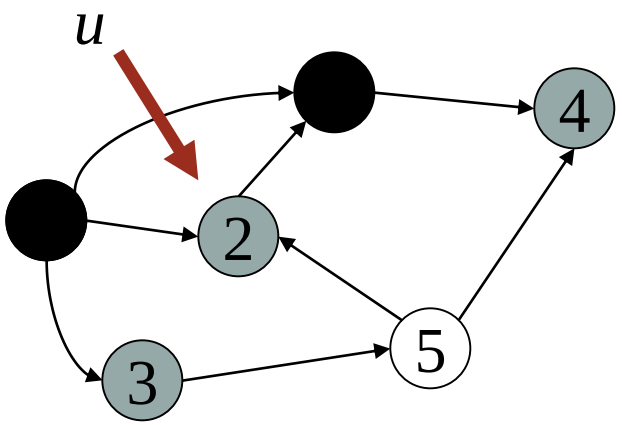
            Enqueue(Q,v)

        }

    Dequeue(Q)

    color[u] ← 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 $Q \neq \varnothing$

{

    $u \leftarrow head[Q];$

    for each $v \in Adjacent[u]$

        if $color[v] = white$

        {

            $color[v] \leftarrow gray$

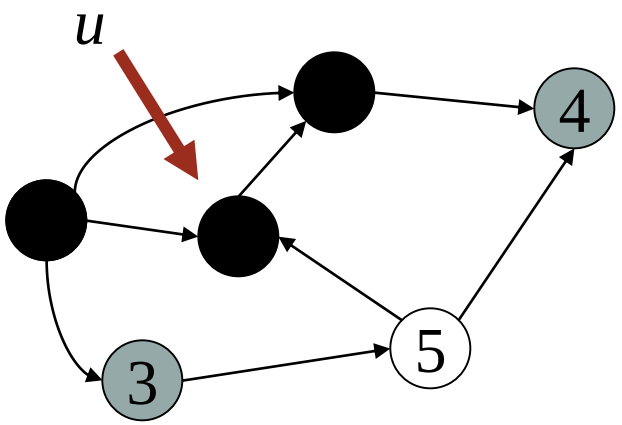
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ | 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 $Q \neq \varnothing$

{

    $u \leftarrow head[Q];$

    for each $v \in Adjacent[u]$

        if $color[v] = white$

        {

            $color[v] \leftarrow gray$

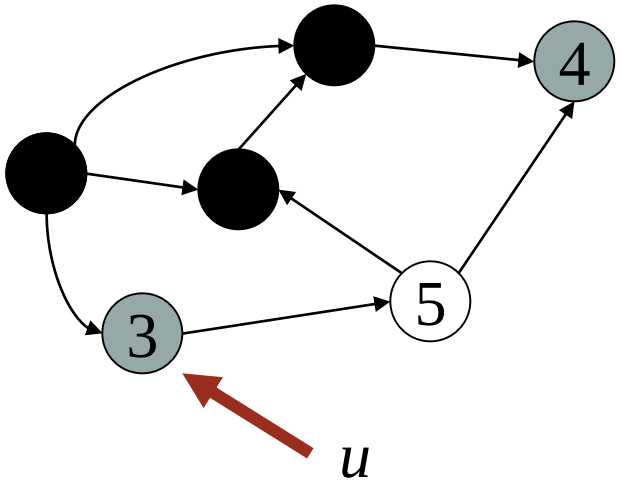
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ | 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 $Q \neq \varnothing$

{

    $u \leftarrow head[Q];$

    for each $v \in Adjacent[u]$

        if $color[v] = white$

        {

            $color[v] \leftarrow gray$

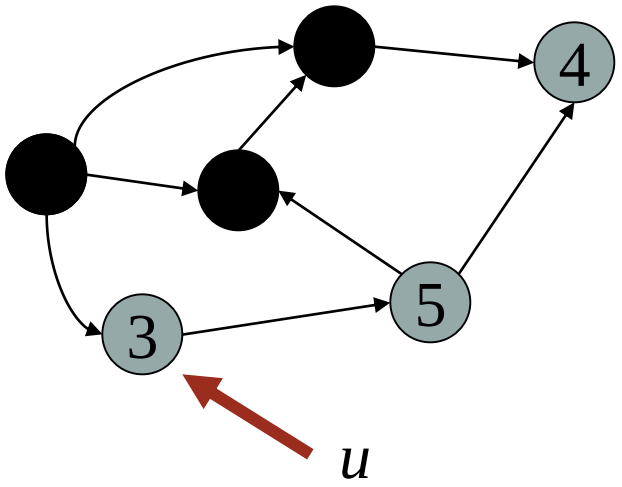
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ | 3 | 4 | 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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

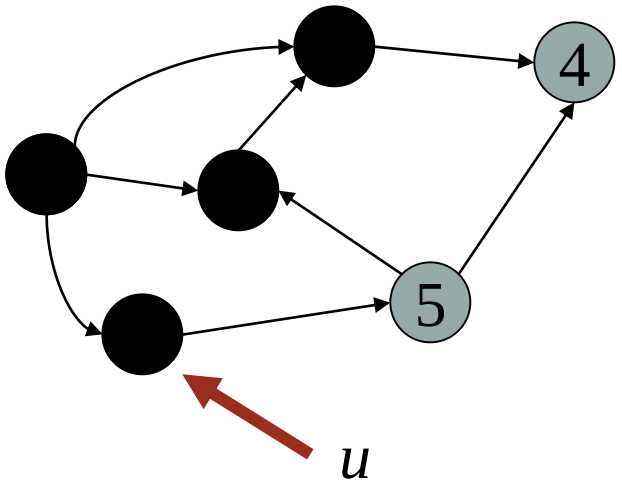
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ | 4 | 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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

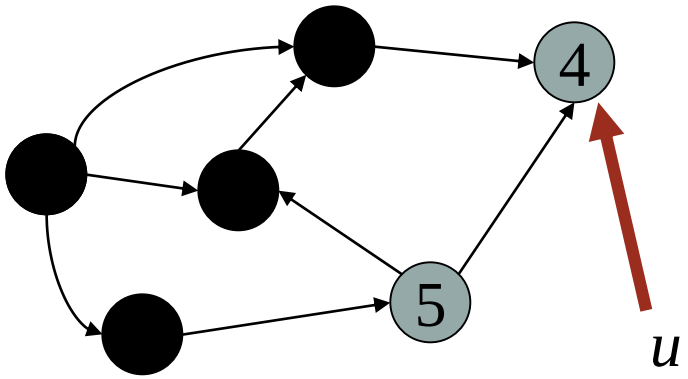
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ | 4 | 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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

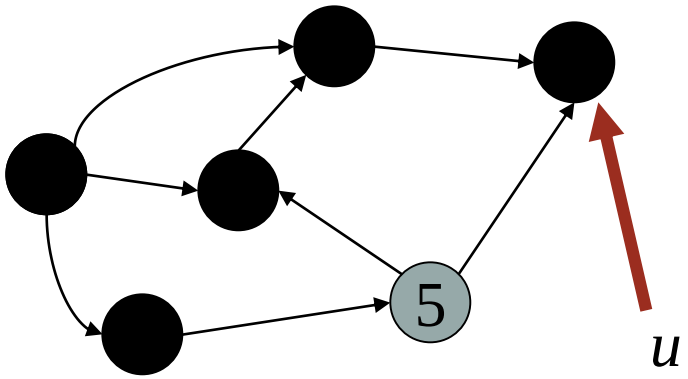
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

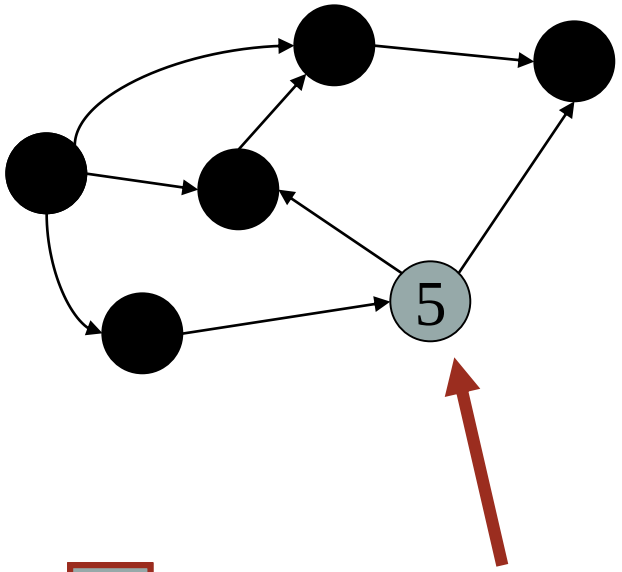
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

}



$Q =$ 5

u

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 $color[v] = white$

        {

            $color[v] \leftarrow gray$

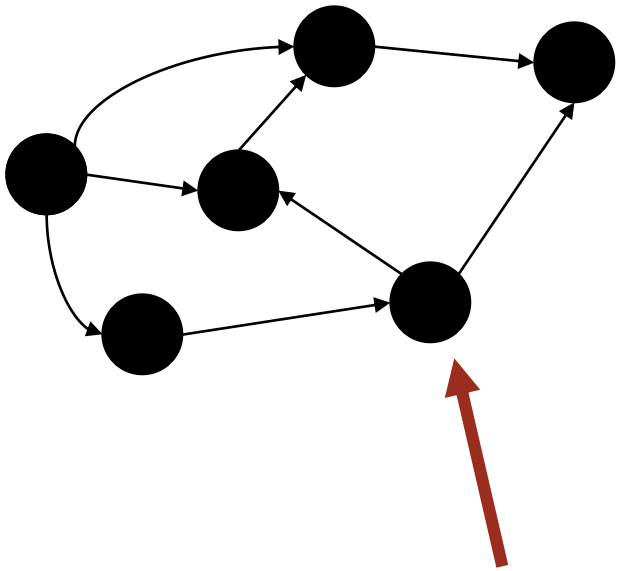
            Enqueue(Q,v)

        }

    Dequeue(Q)

    $color[u] \leftarrow black;$

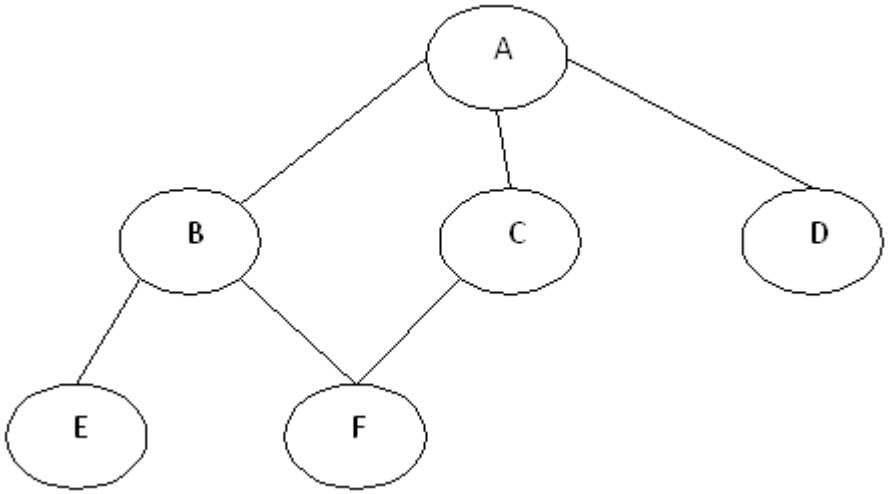
}



$Q = \varnothing$

u

Task source A



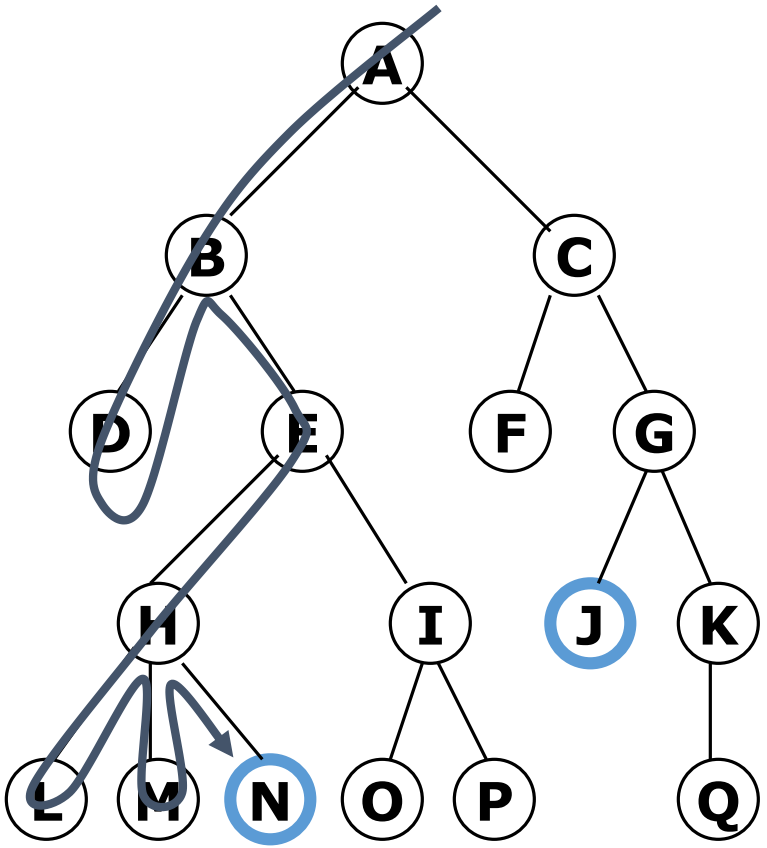
Some Applications of BFS

- Find a shortest path from a vertex s to a vertex v.
- Find the length of such a path.
- Web crawling upto some level

Depth First Search

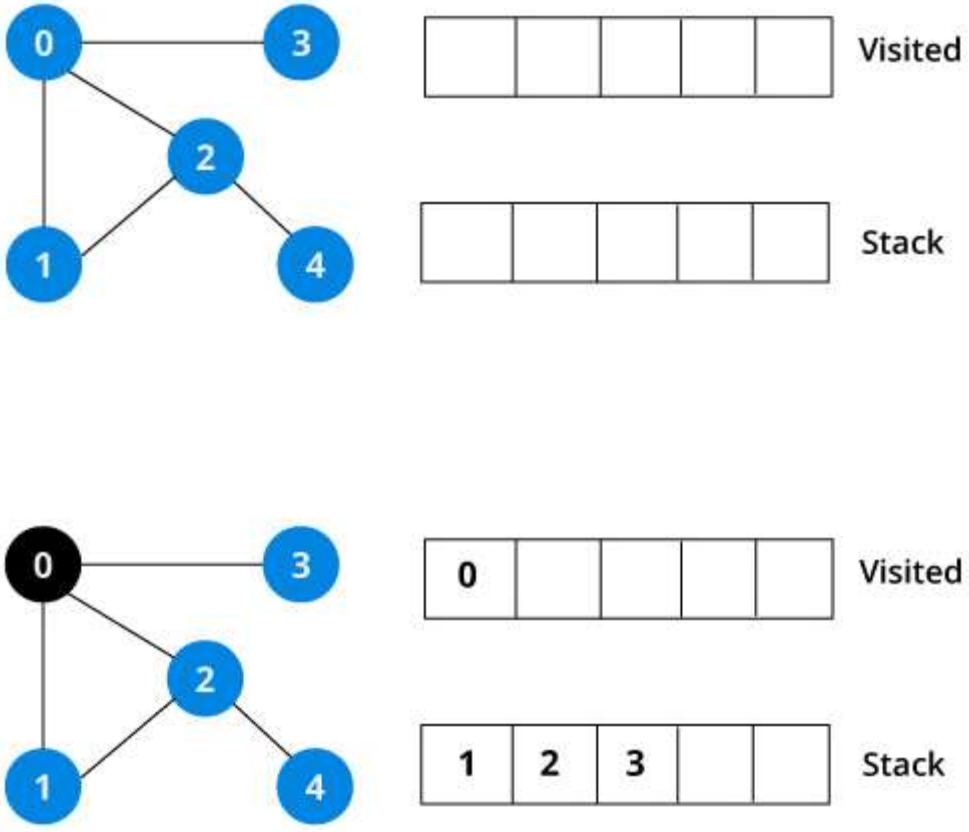
- Depth-first search: Strategy Go as deep as can visiting un-visited nodes
- Choose any un-visited vertex when you have a choice
- When stuck at a dead-end, backtrack as little as possible
- Back up to where you could go to another unvisited vertex
- Then continue to go on from that point
- Eventually you'll return to where you started

Depth-first searching

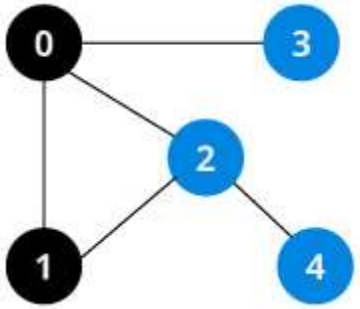


- A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path
- For example, after searching A, then B, then D, the search backtracks and tries another path from B
- Node are explored in the order A B D E H L M N I O P C F G J K Q
- N will be found before J

DFS Example

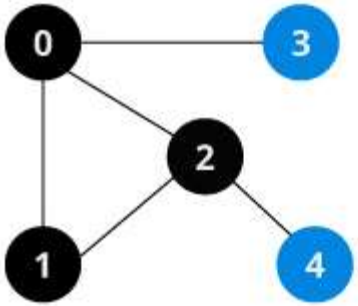


DFS Example



0	1				Visited

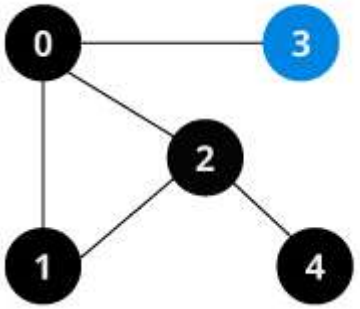
2	3				Stack



0	1	2			Visited

4	3				Stack

DFS Example

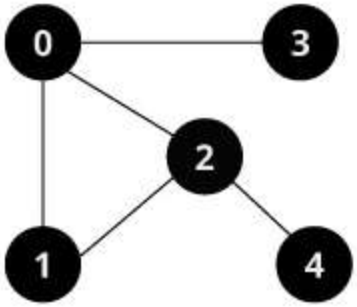


0	1	2	4	

Visited

3				

Stack

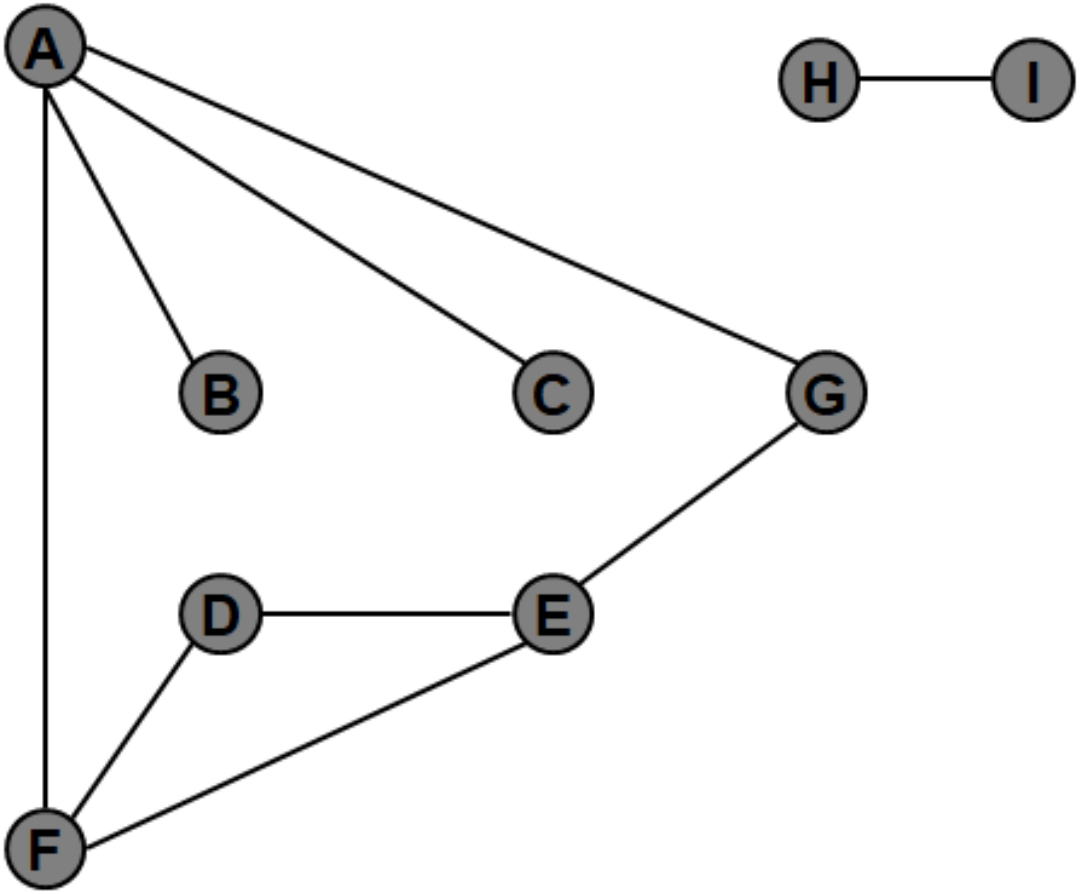


0	1	2	4	3

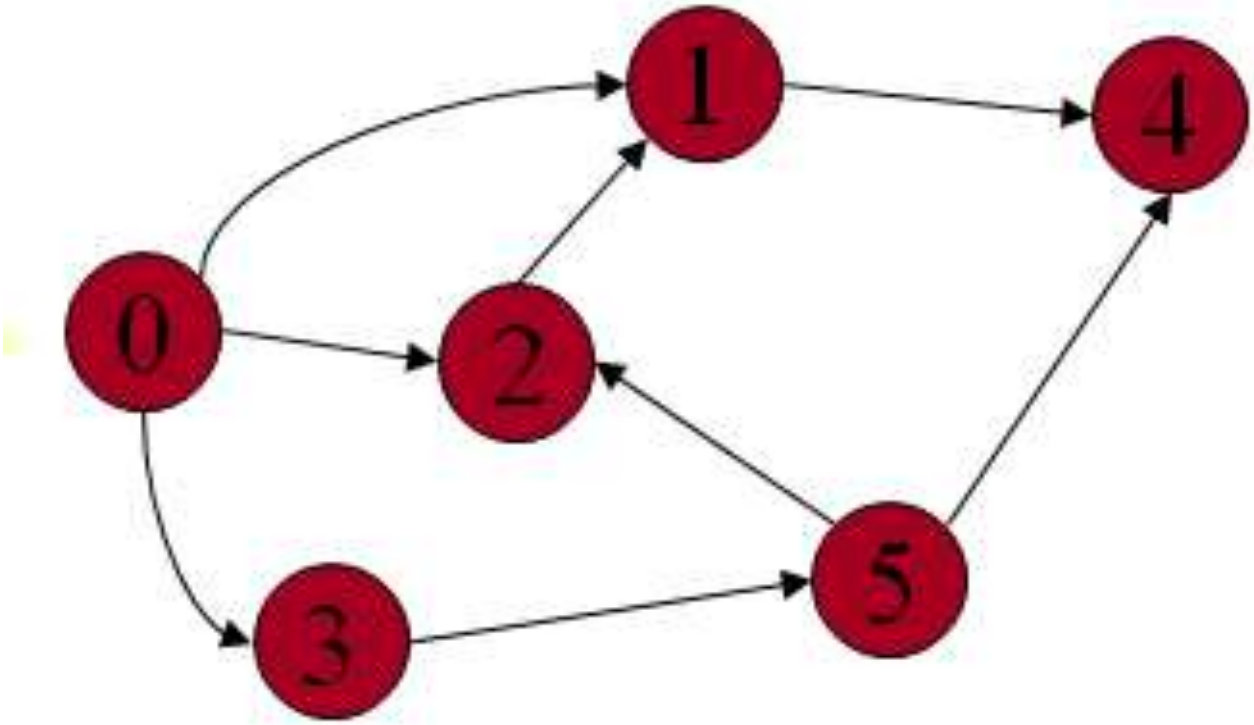
Visited

Stack

Task



Task



Applications of DFS

- Topological sorting
- Finding strongly connected components

Implementation of BFS

```
void bfs(struct Graph* graph, int startVertex) {
   struct queue* q = createQueue();
   graph->visited[startVertex] = 1;
   enqueue(q, startVertex);
    while(!isEmpty(q)){
      printQueue(q);
      int currentVertex = dequeue(q);
      printf("Visited %d\n", currentVertex);
        struct node* temp = graph->adjLists[currentVertex];
        while(temp) {
         int adjVertex = temp->vertex;
         if(graph->visited[adjVertex] == 0){
            graph->visited[adjVertex] = 1;
            enqueue(q, adjVertex);
         }
         temp = temp->next;
     }
   }
}
```

```
struct queue* createQueue() {
    struct queue* q = malloc(sizeof(struct queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

int isEmpty(struct queue* q) {
    if(q->rear == -1)
        return 1;
    else
        return 0;
}
```

```
void enqueue(struct queue* q, int value){
    if(q->rear == SIZE-1)
        printf("\nQueue is Full!!");
    else {
        if(q->front == -1)
            q->front = 0;
        q->rear++;
        q->items[q->rear] = value;
    }
}
```

```
int dequeue(struct queue* q){
    int item;
    if(isEmpty(q)){
        printf("Queue is empty");
        item = -1;
    }
    else{
        item = q->items[q->front];
        q->front++;
        if(q->front > q->rear){
            printf("Resetting queue");
            q->front = q->rear = -1;
        }
    }
    return item;
}
```

Implementation of DFS

```
void DFS(struct Graph* graph, int vertex) {
    struct node* adjList = graph->adjLists[vertex];
    struct node* temp = adjList;
   graph->visited[vertex] = 1;
    printf("Visited %d \n", vertex);
    while(temp!=NULL) {
        int connectedVertex = temp->vertex;
        if(graph->visited[connectedVertex] == 0) {
            DFS(graph, connectedVertex);
        }
        temp = temp->next;
    }
}
```