QUEUE

Algorithm Problem Solving – Samsung Vietnam Mobile R&D Center

Compose by Tran Trung Hieu

hieu.tt3@samsung.com

Contents

Linear Queue

- Queue
- Structure and basic operation
- Example of queue operation
- Queue Implementation

Circular Queue

- Circular Queue
- Example of Circular Queue
- Circular Queue Implementation

Breadth First Search

- BFS (Breadth First Search)
- Example of BFS
- BFS Implementation

Linear Queue

Queue

Characteristic of a queue

A data structure which have the limitation of selection/deletion position like stack.

- Insertions only occur behind the queue, Deletions only occur before the queue.

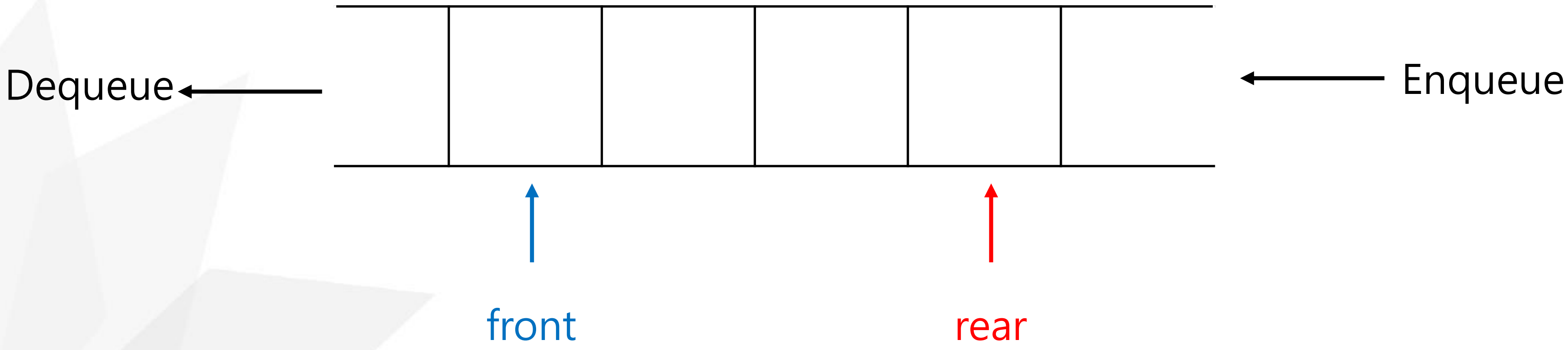
FIFO (First In First Out) structure:

- Elements are saved in order of insertion into the queue.
- Insertion first → get deleted first.

https://www.javascripttutorial.net/javascript-queue/

Structure and Basic Operations

FIFO principle



Structure and Basic Operations

Major Operations of Queue

Operation	Function
front	Get the position of front item from queue
rear	Get the position of rear item from queue
enQueue(item)	An operation to insert elements behind the queue (rear)
deQueue()	Delete and return elements before the queue (front)
createQueue()	Generate a queue of empty state
isEmpty()	Check whether a queue is empty
Qpeek()	Return elements before a queue (front) without delete

Example of Queue Operation

Create an empty queue: createQueue().

Insert element A: enQueue(A).

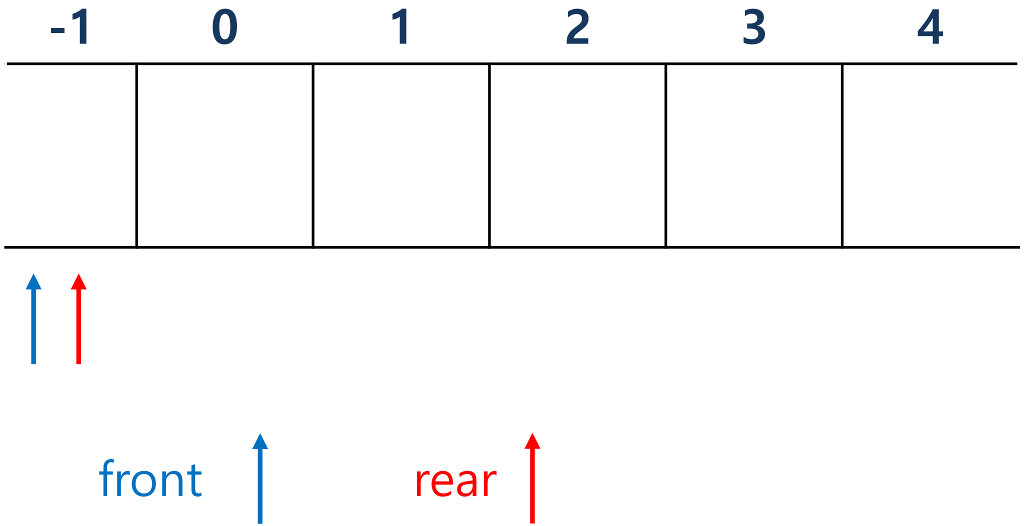
Insert element B: enQueue(B).

Return/Delete element: deQueue().

Insert element C: enQueue(C).

Return/Delete element deQueue()

Return/Delete element deQueue()



Example of Queue Operation

Create an empty queue: createQueue().

Insert element A: enQueue(A).

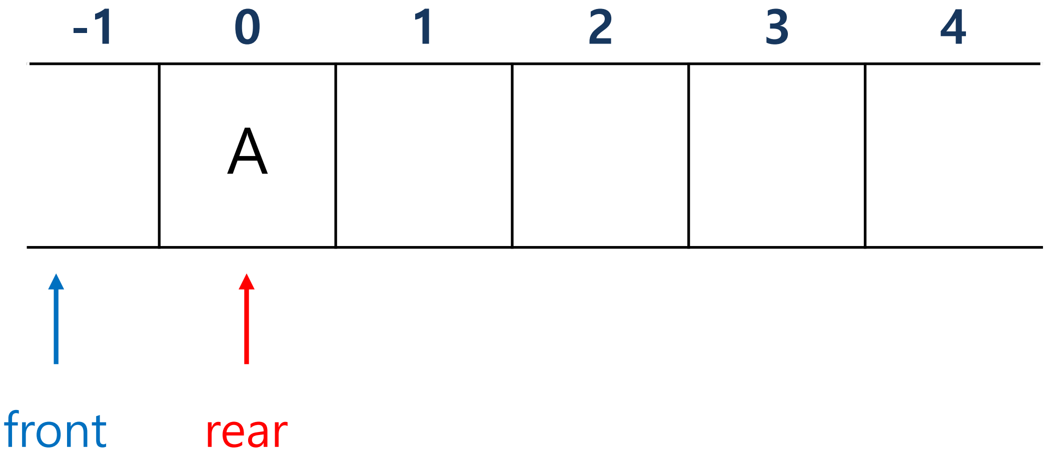
Insert element B: enQueue(B).

Return/Delete element: deQueue().

Insert element C: enQueue(C).

Return/Delete element deQueue()

Return/Delete element deQueue()



-1	0	1	2	3	4
	A				

front

rear

Example of Queue Operation

Create an empty queue: createQueue().

Insert element A: enQueue(A).

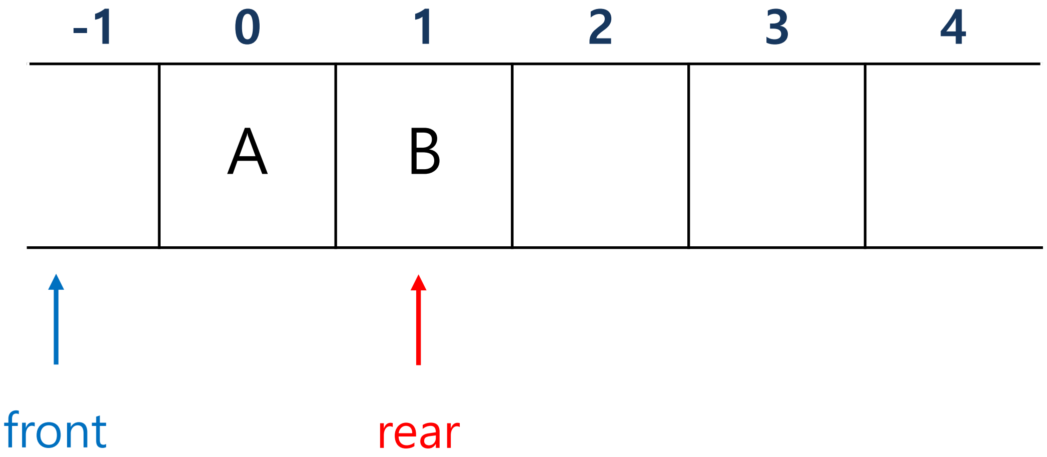
Insert element B: enQueue(B).

Return/Delete element: deQueue().

Insert element C: enQueue(C).

Return/Delete element deQueue()

Return/Delete element deQueue()



-1	0	1	2	3	4
	A	B			

Example of Queue Operation

Create an empty queue: createQueue().

Insert element A: enQueue(A).

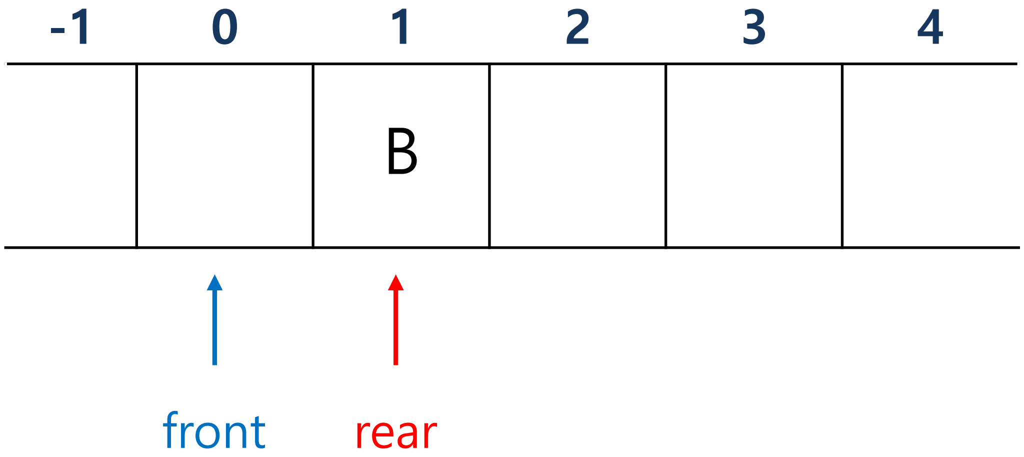
Insert element B: enQueue(B).

Return/Delete element: deQueue().

Insert element C: enQueue(C).

Return/Delete element deQueue()

Return/Delete element deQueue()



Example of Queue Operation

Create an empty queue: createQueue().

Insert element A: enQueue(A).

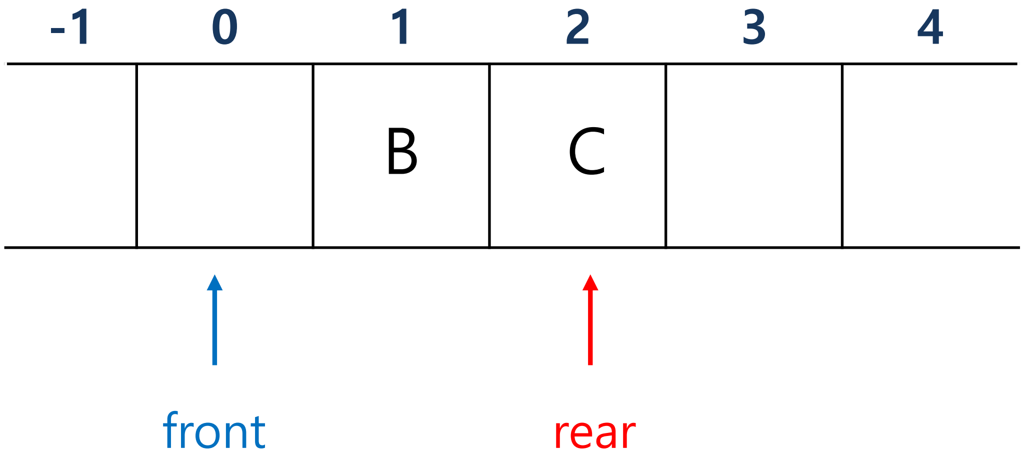
Insert element B: enQueue(B).

Return/Delete element: deQueue().

Insert element C: enQueue(C).

Return/Delete element deQueue()

Return/Delete element deQueue()



-1	0	1	2	3	4
		B	C		

front

rear

Example of Queue Operation

Create an empty queue: createQueue().

Insert element A: enQueue(A).

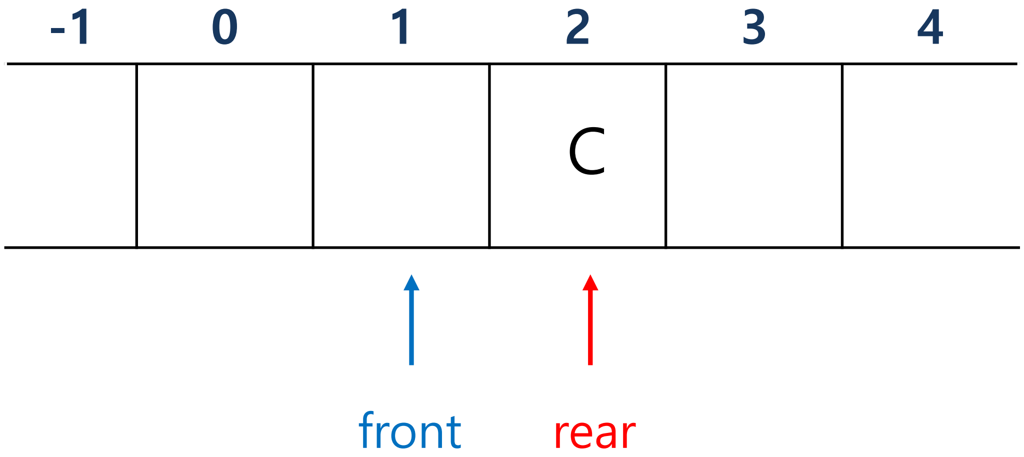
Insert element B: enQueue(B).

Return/Delete element: deQueue().

Insert element C: enQueue(C).

Return/Delete element deQueue().

Return/Delete element deQueue().



Example of Queue Operation

Create an empty queue: createQueue().

Insert element A: enQueue(A).

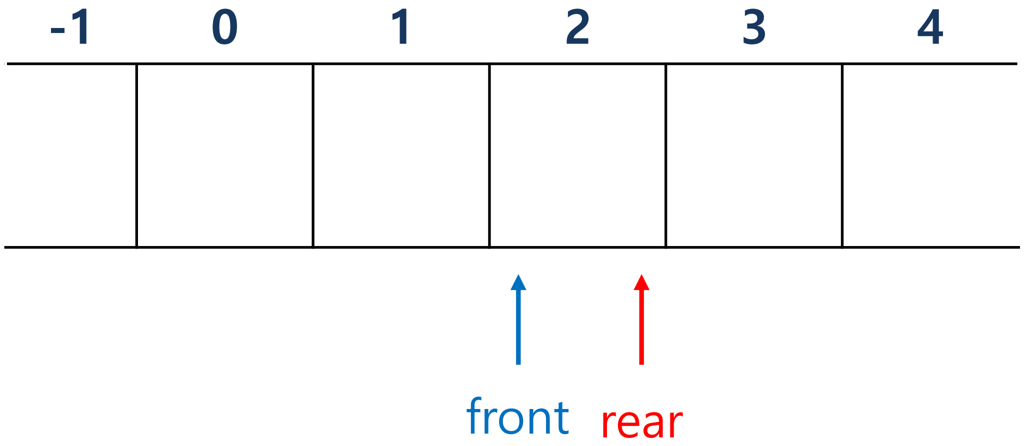
Insert element B: enQueue(B).

Return/Delete element: deQueue().

Insert element C: enQueue(C).

Return/Delete element deQueue().

Return/Delete element deQueue().



Queue Implementation

Linear Queue

A queue use a 1D – array

- Queue size = array size (n)
- Front: index of the first saved element
- Rear: index of the last saved element

State Expression

- Initial state: front = rear = -1
- Empty state: front = rear
- Full state: rear = n-1 (n: array size)

Generate a initial empty queue

- Generate 1D – array of size N.
- Initialize front and rear to -1.

Queue Implementation

Linear Queue

Insert new element behind the queue

```
Algorithm enQueue(Q[N], front, rear, item) {
    if(rear == N-1) Print(“Full  Queue”);
    else {
        rear = rear + 1;
        Q[rear] = item;
    }
}
```

Queue Implementation

Linear Queue

Delete element in front of the queue

```
Algorithm deQueue(Q[N], front, rear) {
    if(rear == front) Print(“Empty queue”);
    else {
        front = front + 1;
        return Q[front];
    }
}
```

Queue Implementation

Linear Queue

Check empty queue

```
Algorithm isEmpty(Q[N], front, rear, item) {
    if(rear == front) Print(“Empty Queue”);
}
```

Check full queue

```
Algorithm isFull(Q[N], front, rear, item) {
    if(rear == N-1) Print(“Full Queue”);
}
```

Queue Implementation

Linear Queue

Get front of queue

```
Algorithm Qpeek(Q[N], front, rear, item) {
    if(rear == -1 && front == -1) Print(“Empty Queue”);
    return Q[front+1];
}
```

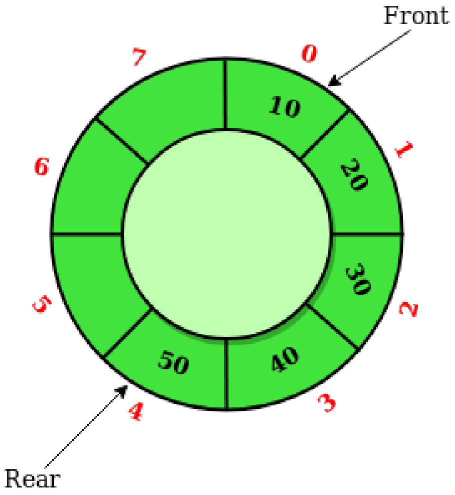
Circular Queue

Circular Queue Structure

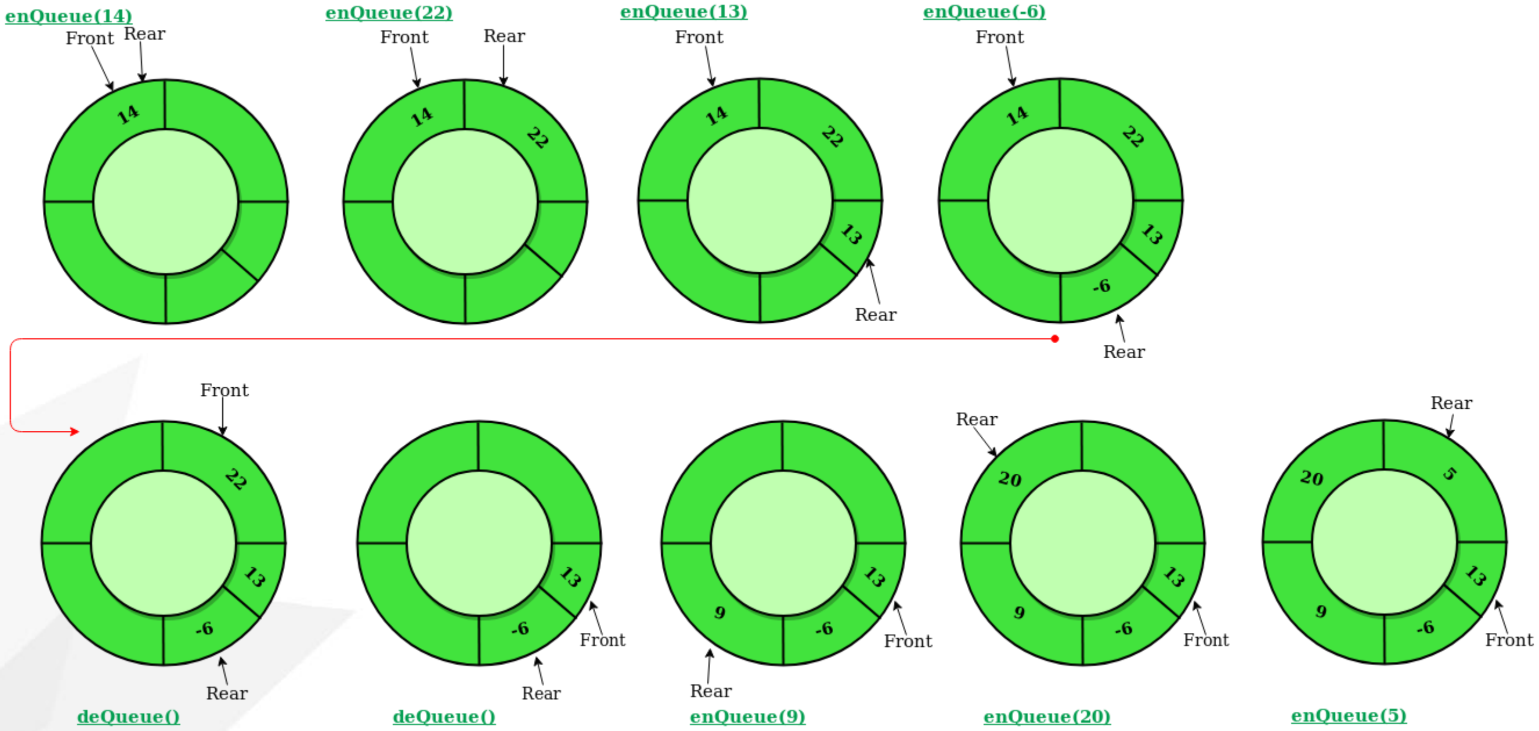
‘Ring buffer’ structure

Circular queue, also called ‘Ring buffer’, is a linear data structure in which the operation are performed based on FIFO principle and the last position is connected back to the first position to make a circle.

Circular queue has similar operation with basic queue.



Example of Circular Queue Operations



Circular Queue Implementation

Insert new element

```
Algorithm enQueue(Q[N], front, rear, item) {
    if((front == -1 && rear == N-1) ||
           rear == front-1) Print(“Full  Queue”);
    else if(rear == N-1 && front != 0) {
        rear = 0;
        Q[rear]  = item;
    }
    else {
        rear = rear + 1;
        Q[rear] = item;
    }
}
```

Circular Queue Implementation

Delete element in front of the queue

```
Algorithm deQueue(Q[N], front, rear) {
    if(front == -1 && rear == -1) Print(“Empty queue”);
    front = front + 1;
    data = Q[front];
    if(front == rear){
        front = rear = -1;
    }
    else if(front == N-1){
        front = -1;
    }
    return data;
}
```

Breadth First Search

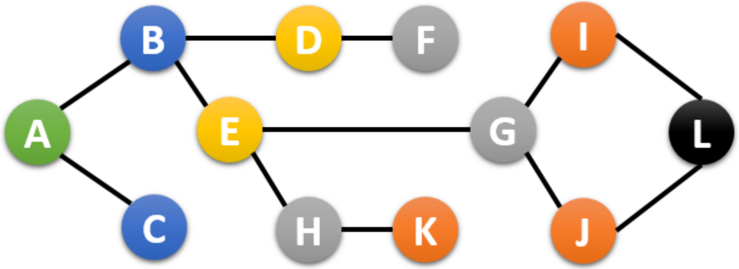
BFS (Breadth First Search)

Graph or Tree traversing algorithm

- Start at “root” node, and explore all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

- If current node is not destination, continue to explore all of the neighbor.

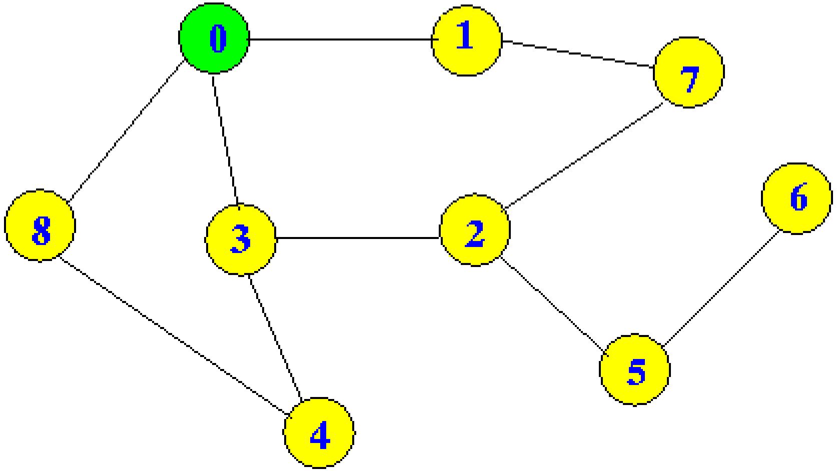
- The algorithm finish when destination is found/all nodes are visited.



Breadth-First Search (BFS)

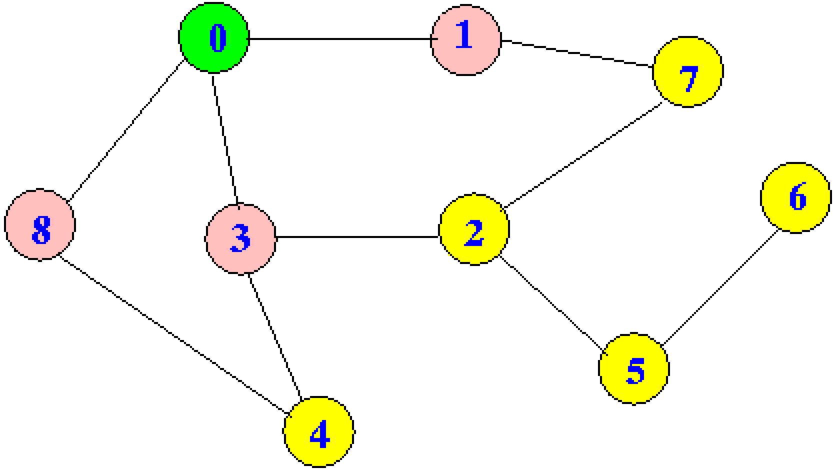
Example of BFS

- Start at some nodes (e.g.,, Node 0)



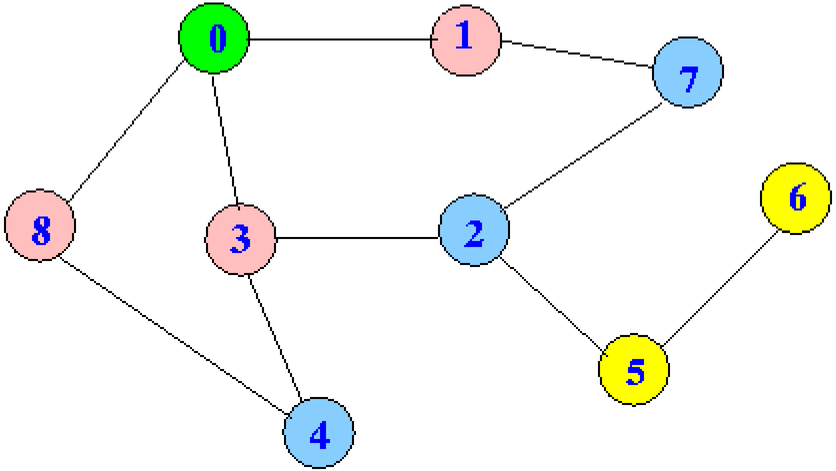
Example of BFS

- Visit **all the neighbor** of **Node 0** first



Example of BFS

Then visit the **neighbors of neighbors**



BFS Implementation

The BFS Algorithm is implemented by:

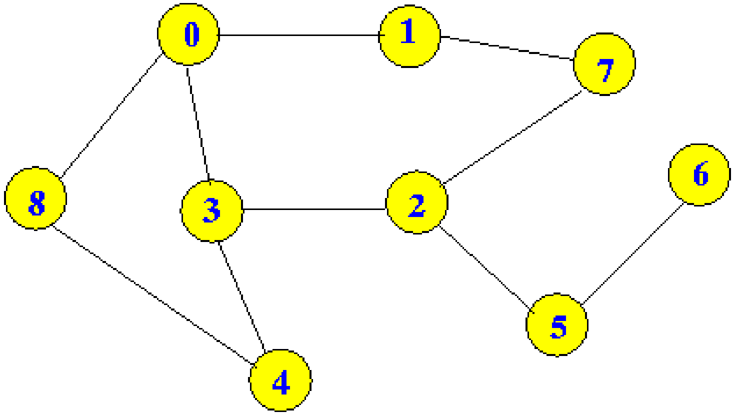
Using a queue to store the node in the list of the node will visit.

Pseudo code:

```
Set all nodes to "not visited";

 q = new Queue();

 q.enqueue(initial node);

 while ( q ≠ empty ) do
 {
    x = q.dequeue();

    if ( x has not been visited )
    {
       visited[x] = true;        // Visit node x !

       for ( every edge (x, y)  /* we are using all edges ! */ )
          if ( y has not been visited )
                                 q.enqueue(y);       // Use the edge (x,y) !!!
    }
 }
```

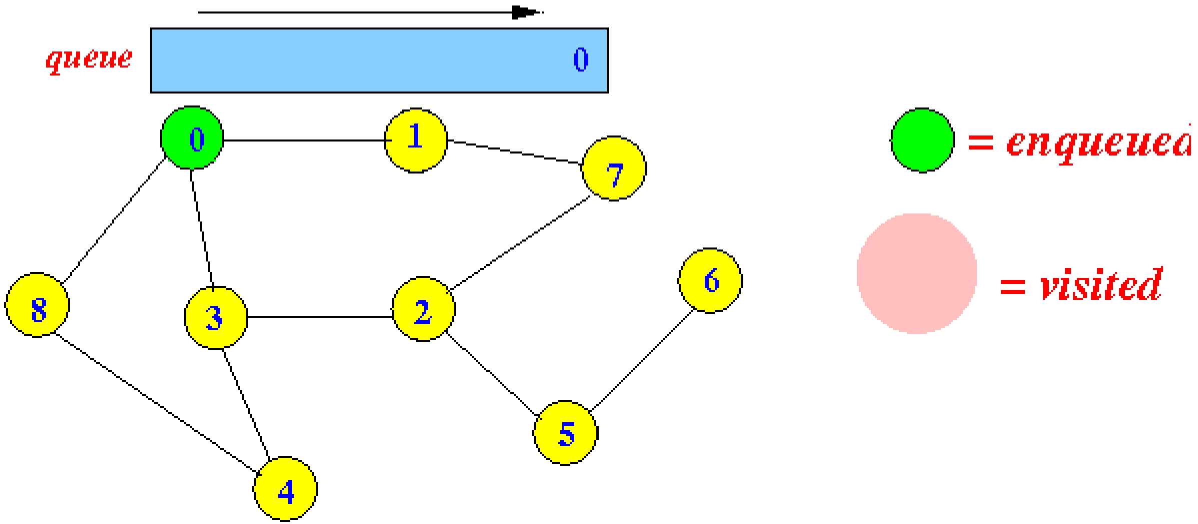
BFS Implementation

There is a graph



BFS Implementation

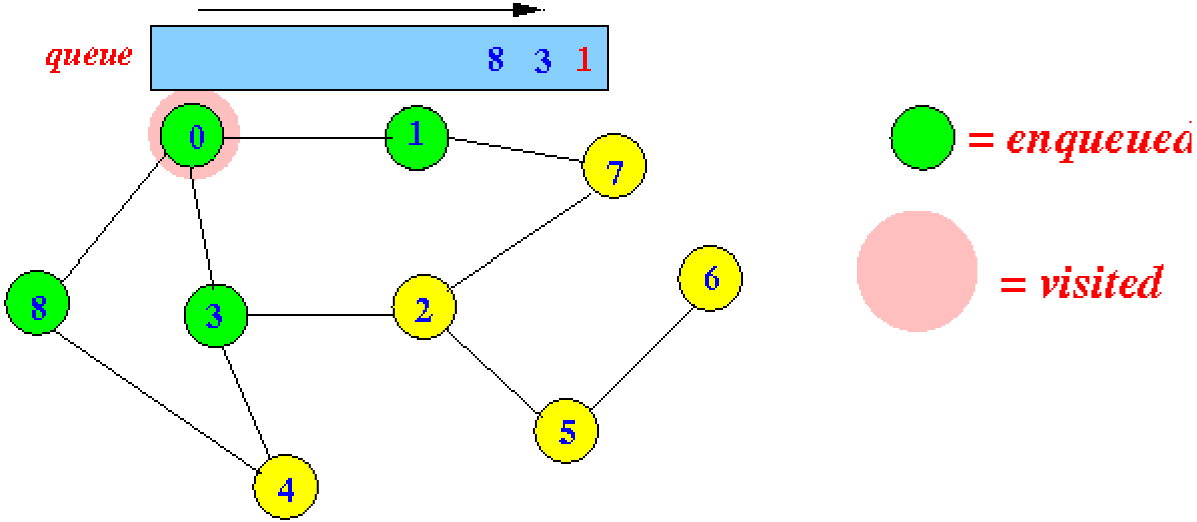
Initial state: node 0 is enqueued



BFS Implementation

After visit node 0, node 1, 3, 8 is enqueued.

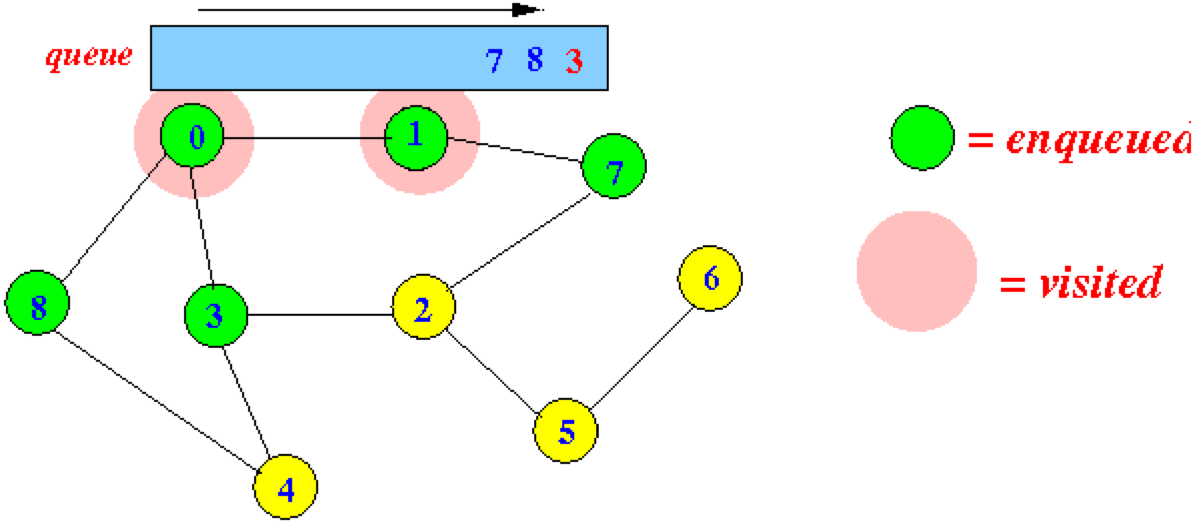
Next step: visit node 1



BFS Implementation

After visit node 1, node 7 is enqueued.

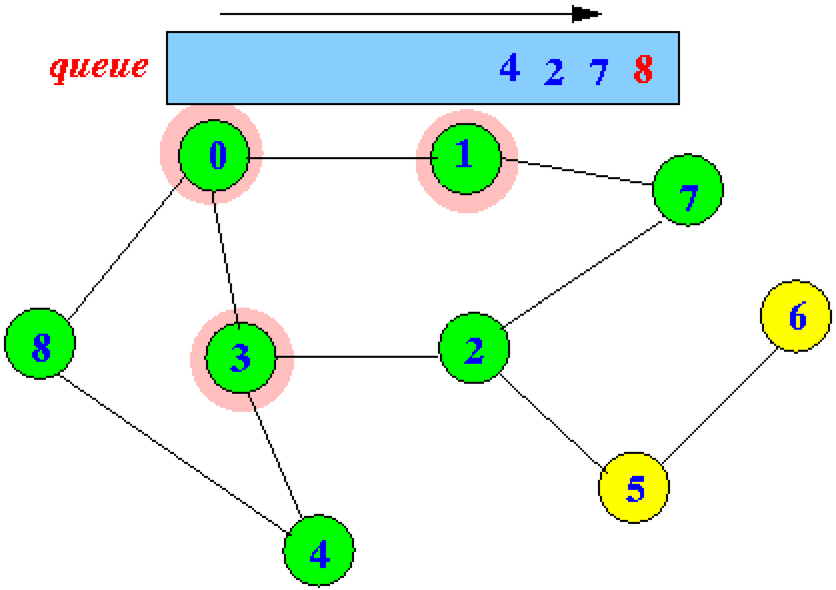
Next step: visit node 3



BFS Implementation

After visit node 3, node 4 is enqueued.

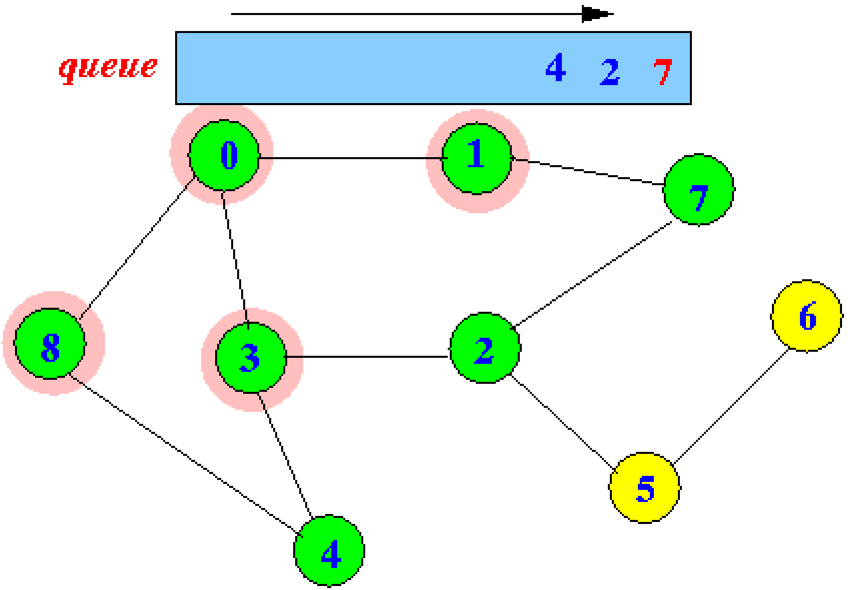
Next step: visit node 8



BFS Implementation

After visit node 8, nothings is enqueued.

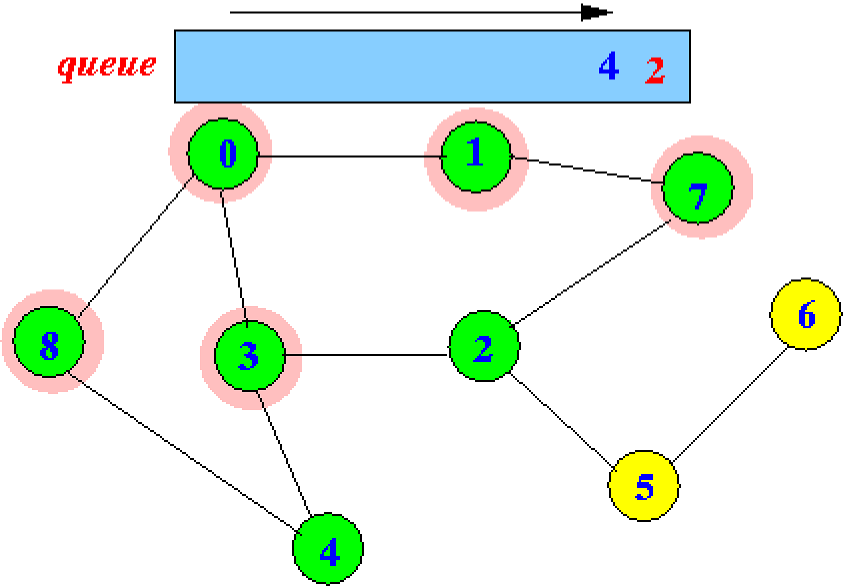
Next step: visit node 7



BFS Implementation

After visit node 7, nothings is enqueued.

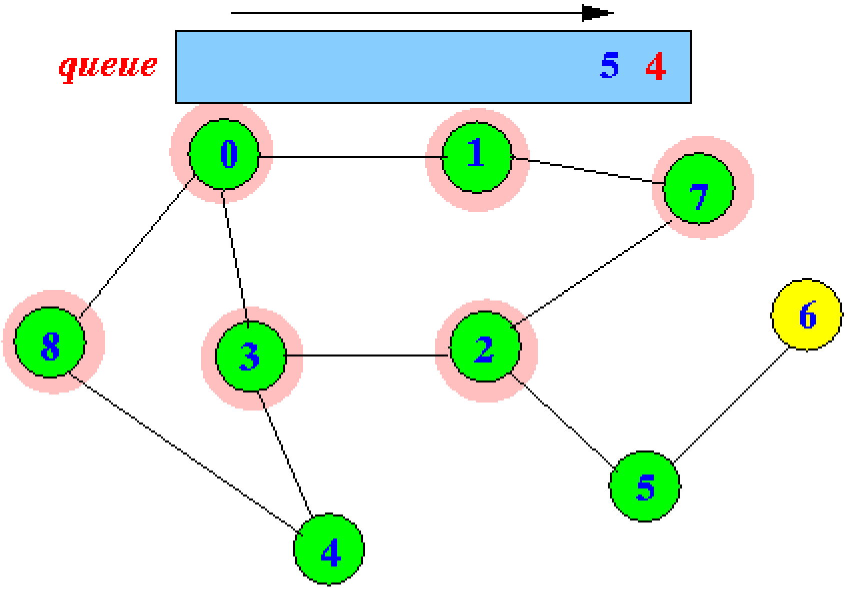
Next step: visit node 2



BFS Implementation

After visit node 2, node 5 is enqueued.

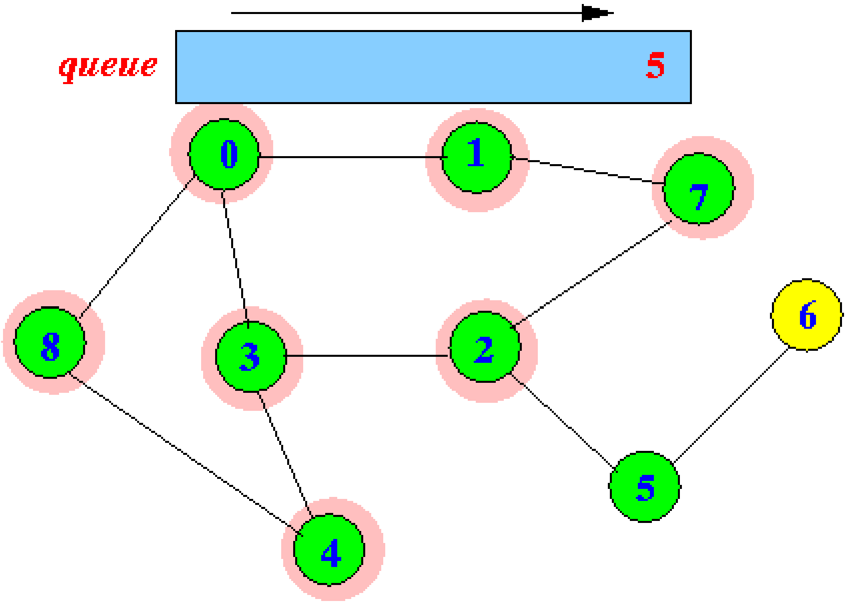
Next step: visit node 4



BFS Implementation

After visit node 4, nothings is enqueued.

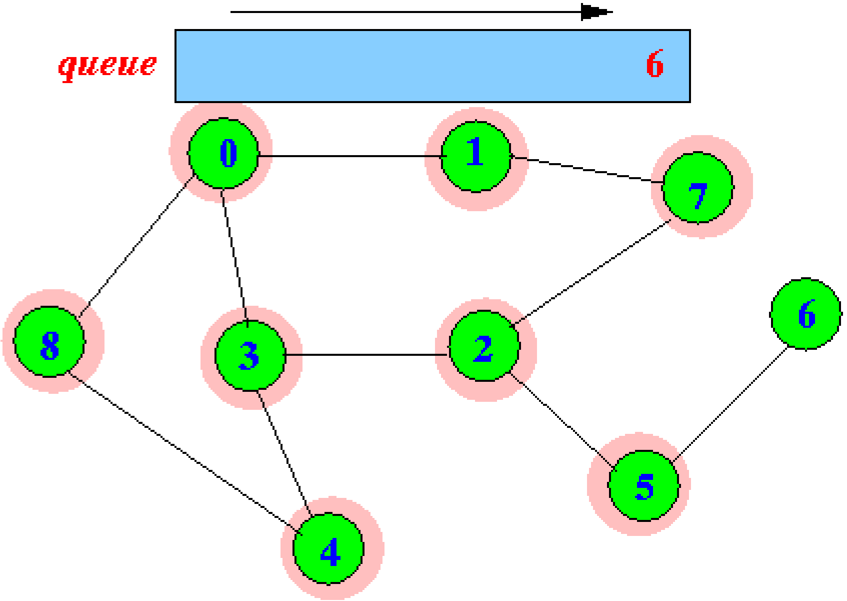
Next step: visit node 5



BFS Implementation

After visit node 5, node 6 is enqueued.

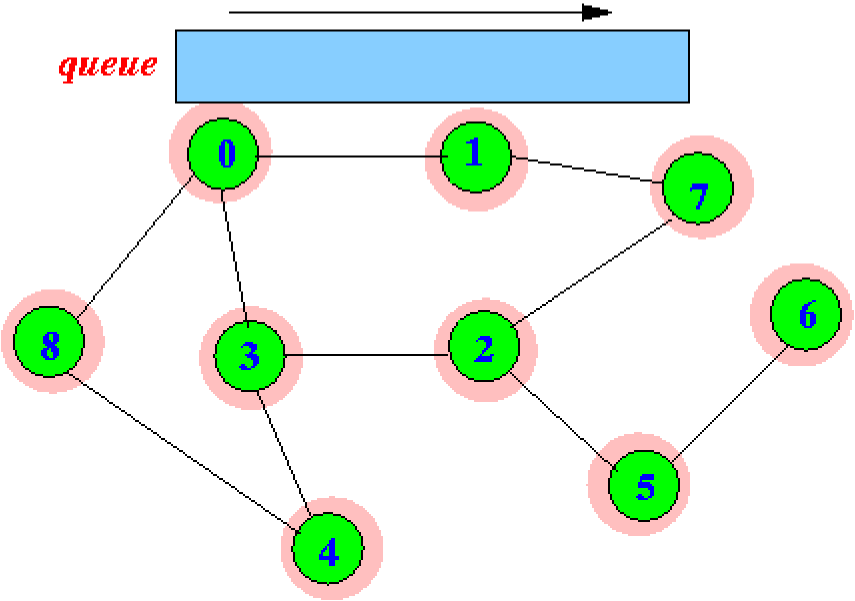
Next step: visit node 6



BFS Implementation

After visit node 6, nothings is enqueued.

Queue is be empty. All nodes are visited → DONE



Reference

https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/

http://alexvolov.com/2015/02/breadth-first-search-bfs/

http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/bfs.html

Thank you!