

Queues

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2025



Outline

- 1 Definition
- 2 Implementation
- 3 Sequential Queue & Circular Queue

Outline

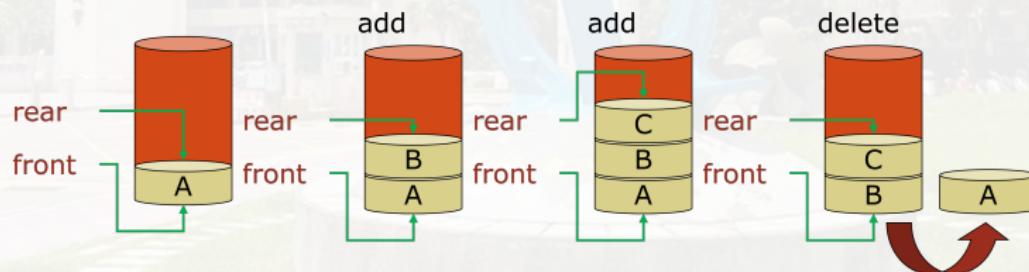
1 Definition

2 Implementation

3 Sequential Queue & Circular Queue

Definition

- A queue is an ordered list in which **insertions** take place at one end (i.e., **front**) and deletions take place at the opposite end (i.e., **rear**).
 - insertions: push/add
 - deletions: pop/remove
- First-In-First-Out (FIFO).



Outline

1 Definition

2 Implementation

3 Sequential Queue & Circular Queue

Functions for Queues

- Create a queue (implemented by an **array**).
 - Create an empty queue with maximum size MAX_QUEUE_SIZE.

```
#define MAX_QUEUE_SIZE 100

typedef struct {
    int key; // can be of other types...
    /* other fields? */
} element;

element queue a[MAX_QUEUE_SIZE];
int front = -1; // initially no element
int rear = -1; // initially no element
```

Functions for Queues (2/2)

- IsEmpty
 - Return true if the queue is empty and false otherwise.

Functions for Queues (2/2)

- IsEmpty
 - Return true if the queue is empty and false otherwise.
`front == rear;`
- IsFull
 - Return true if the queue is full and false otherwise.

Functions for Queues (2/2)

- IsEmpty
 - Return true if the queue is empty and false otherwise.
`front == rear;`
- IsFull
 - Return true if the queue is full and false otherwise.
`rear == MAX_QUEUE_SIZE-1;`
- Enqueue (or AddQ)
 - Insert the element into the `rear` of the queue.

Functions for Queues (2/2)

- IsEmpty
 - Return true if the queue is empty and false otherwise.
`front == rear;`
- IsFull
 - Return true if the queue is full and false otherwise.
`rear == MAX_QUEUE_SIZE-1;`
- Enqueue (or AddQ)
 - Insert the element into the `rear` of the queue.
If the queue is not full, `queue[++rear] = element;`
- Dequeue (or DeleteQ)
 - Remove and return the item at the front of the queue.

Functions for Queues (2/2)

- IsEmpty
 - Return true if the queue is empty and false otherwise.
`front == rear;`
- IsFull
 - Return true if the queue is full and false otherwise.
`rear == MAX_QUEUE_SIZE-1;`
- Enqueue (or AddQ)
 - Insert the element into the `rear` of the queue.
If the queue is not full, `queue[++rear] = element;`
- Dequeue (or DeleteQ)
 - Remove and return the item at the front of the queue.
If the queue is not empty, `return stack[front++];`

addQ or Enqueue

```
void Enqueue(element item) { // add item to the queue
    if (rear == MAX_QUEUE_SIZE - 1) {
        queueFull();
    }
    queue[++rear] = item;
}

void queueFull() {
    fprintf(stderr, "Queue is FULL!!");
    exit(EXIT_FAILURE);
}
```

delteQ or Dequeue

```
element Dequeue() { // no argument is required!
    if (front == rear) {
        return queueEmpty(); // return an error
    }
    return queue[front++];
}

element queueEmpty() {
    element errKey; // depending the struct of element
    errKey.key = -99;
    fprintf(stderr, "Queue is EMPTY!!!");
    return errorKey;
}
```

Outline

- 1 Definition
- 2 Implementation
- 3 Sequential Queue & Circular Queue

Job Scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	comments
-1	-1					queue is empty
-1	0	J_1				Job J_1 is added
-1	1	J_1	J_2			Job J_2 is added
-1	2	J_1	J_2	J_3		Job J_3 is added
0	2		J_2	J_3		Job J_1 is deleted
1	2			J_3		Job J_2 is deleted

Job Scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	comments
-1	-1					queue is empty
-1	0	J_1				Job J_1 is added
-1	1	J_1	J_2			Job J_2 is added
-1	2	J_1	J_2	J_3		Job J_3 is added
0	2		J_2	J_3		Job J_1 is deleted
1	2			J_3		Job J_2 is deleted

- If `rear == MAX_QUEUE_SIZE-1`, one suggests that the queue is full (but it's not).

Job Scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	comments
-1	-1					queue is empty
-1	0	J_1				Job J_1 is added
-1	1	J_1	J_2			Job J_2 is added
-1	2	J_1	J_2	J_3		Job J_3 is added
0	2		J_2	J_3		Job J_1 is deleted
1	2			J_3		Job J_2 is deleted

- If `rear == MAX_QUEUE_SIZE-1`, one suggests that the queue is full (but it's not).
- We should move the ENTIRE queue to the left.

Job Scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	comments
-1	-1					queue is empty
-1	0	J_1				Job J_1 is added
-1	1	J_1	J_2			Job J_2 is added
-1	2	J_1	J_2	J_3		Job J_3 is added
0	2		J_2	J_3		Job J_1 is deleted
1	2			J_3		Job J_2 is deleted

- If `rear == MAX_QUEUE_SIZE-1`, one suggests that the queue is full (but it's not).
- We should move the ENTIRE queue to the left. $\Rightarrow O(MAX_QUEUE_SIZE)$ (very time consuming!)

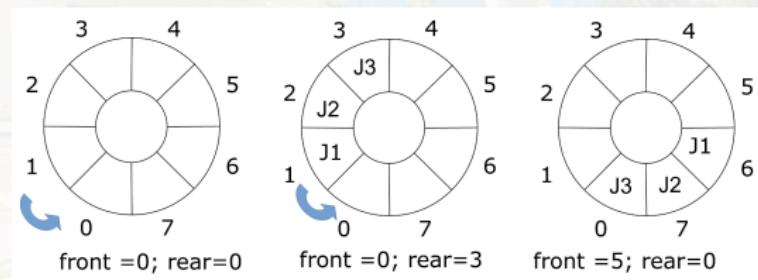


Solution: Circular Queue

- Initially, `front = rear = 0;`

Solution: Circular Queue

- Initially, $\text{front} = \text{rear} = 0$;
- front : one position counterclockwise from the first element in the queue.
- rear : current end of the queue.



Circular Queue (2/2)

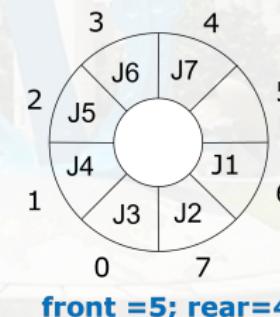
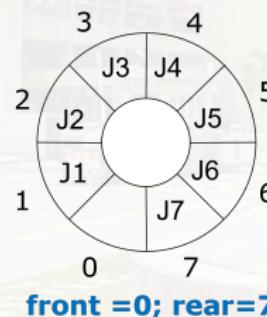
- Such a circular queue is permitted to hold at most elements.

Circular Queue (2/2)

- Such a circular queue is permitted to hold at most `MAX_QUEUE_SIZE - 1` elements.

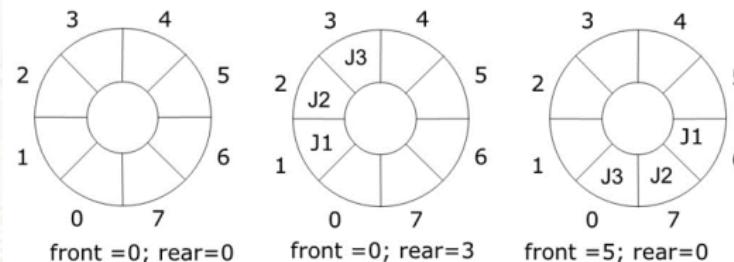
Circular Queue (2/2)

- Such a circular queue is permitted to hold at most `MAX_QUEUE_SIZE - 1` elements.
- The addition of an element such that `front == rear`: the queue is empty (?) or full (?).



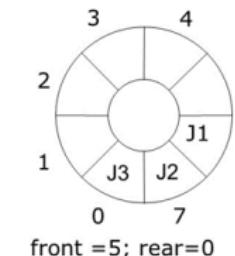
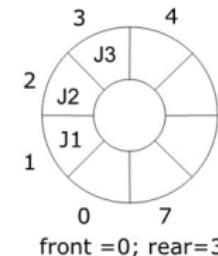
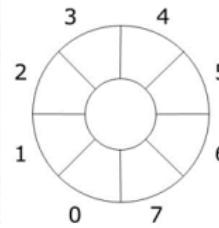
Adding an Element to a Circular Queue

```
void Enqueue(element item) {  
    rear = (rear+1) % MAX_QUEUE_SIZE;  
    if (front == rear) {  
        queueFull(); // reset rear and print error!  
    }  
    queue[rear] = item;  
}
```



Deleting an Element from a Circular Queue

```
element Dequeue() {  
    element item;  
    if (front == rear) {  
        return queueEmpty();  
    }  
    front = (front+1) % MAX_QUEUE_SIZE;  
    return queue[front];  
}
```



Discussions