Queues

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

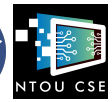
1. Definition
2. Implementation
3. Sequential Queue & Circular Queue

Outline

1 Definition

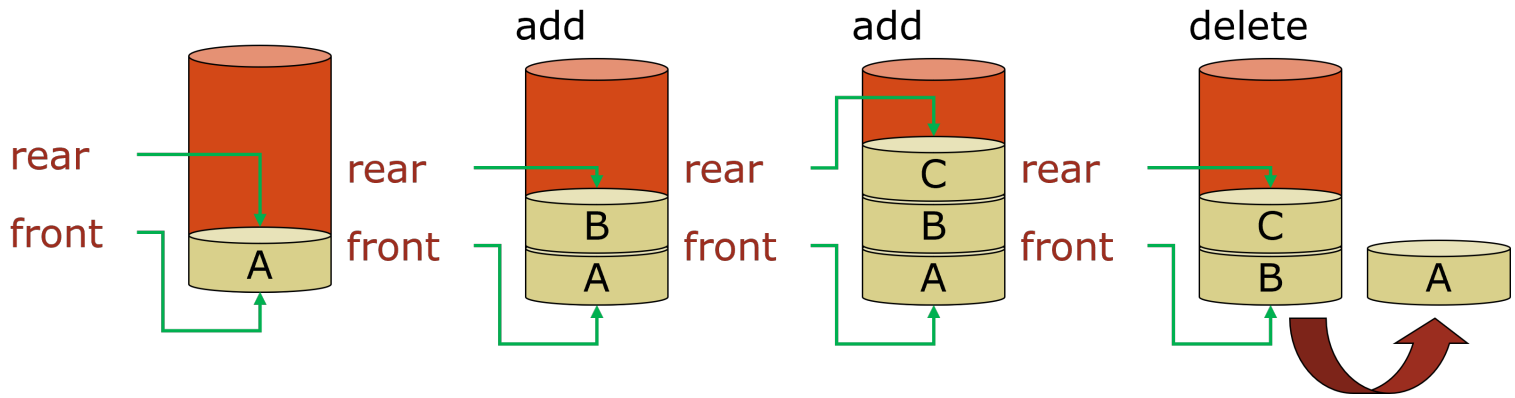
2 Implementation

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 front = -1; // initially no element
```

Functions for Stacks (2/2)

- IsEmpty
  - Return TRUE if the queue is empty and FALSE otherwise.

Functions for Stacks (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 Stacks (2/2)

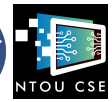
- IsEmpty
  - Return TRUE if the queue is empty and FALSE otherwise.
    `front == rear;`
- IsFull
  - Return TRUE if the queue is full and FALSE otherwise.
    `rear == MAX_QUEUE_SIZE-1`
- Push (or Add)
  - Insert the element into the rear of the queue.

Functions for Stacks (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`
- Push (or Add)
  - Insert the element into the rear of the queue.
    If the queue is not full, `queue[++rear] = element;`
- Pop (or Delete)
  - Remove and return the item at the front of the queue.

Functions for Stacks (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`
- Push (or Add)
  - Insert the element into the rear of the queue.
    If the queue is not full, `queue[++rear] = element;`
- Pop (or Delete)
  - Remove and return the item at the front of the queue.
    If the queue is not empty, `return stack[++front];`

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

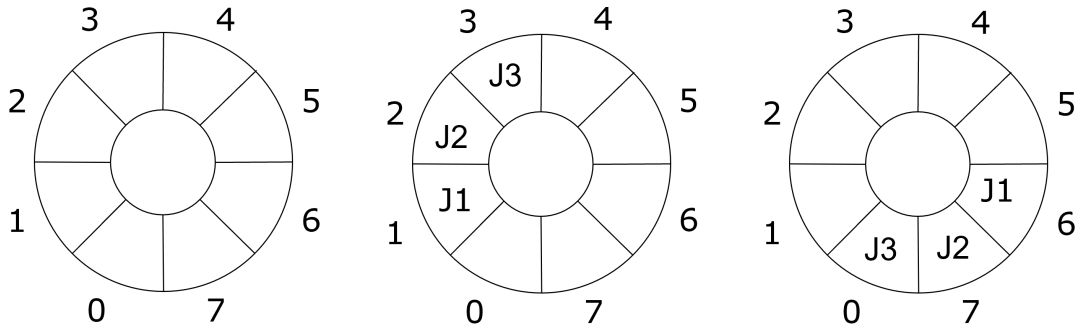
- If `rear == MAX_QUEUE_SIZE-1`, one suggests that the queue is full (but it's not).
- We should move the ENTIRE queue to the left. $\Rightarrow$ $O(MAX\_QUEUE\_SIZE)$ (very time consuming!)

Solution: Circular Queue

- Initially, `front = rear = 0;`

Solution: Circular Queue

- Initially, `front = rear = 0;`
- `front`: one position counterclockwise from the first element in the queue.
- `rear`: current end of the queue.



Circular Queue (2/2)

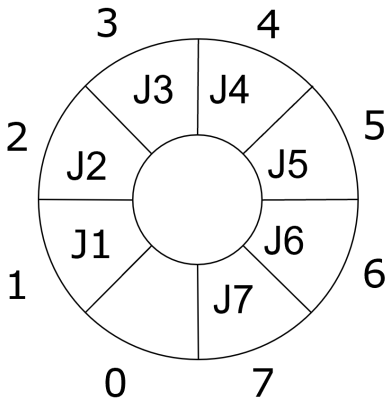
- Such a circular queue is permitted to hold at most elements.

Circular Queue (2/2)

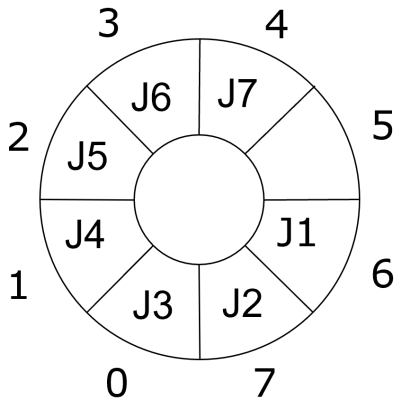
- Such a circular queue is permitted to hold at most MAX_QUEUE_SIZE-1 elements.

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 (?).



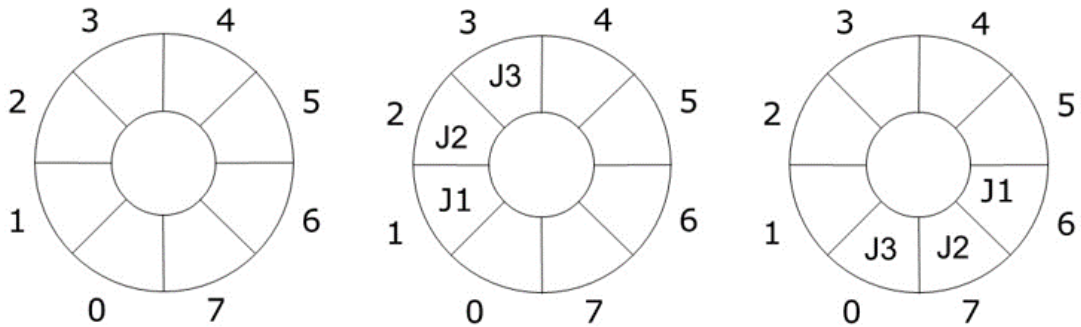
front =0; rear=7



front =5; rear=4

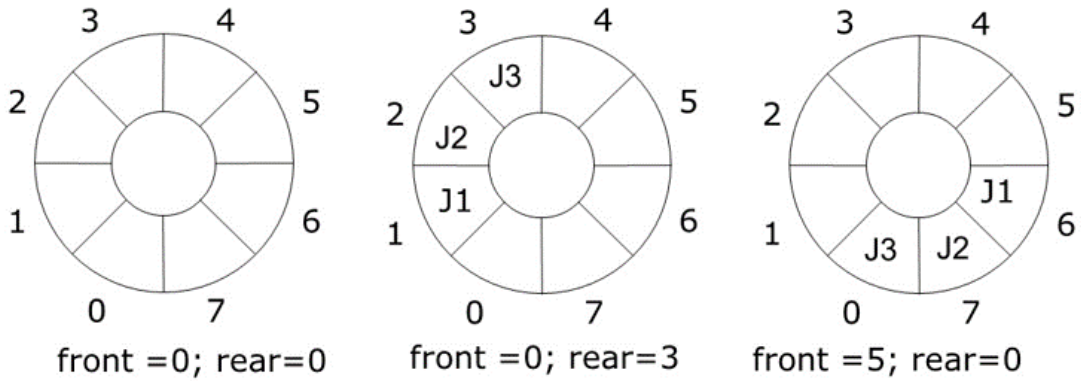
Adding an Element to a Circular Queue

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



Deleting an Element from a Circular Queue

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



Discussions