DATA STRUCTURES AND ALGORITHMS

Circular Queue Data Structure

By
Zainab Malik

Content

- Limitation of Linear Queue
- Introduction to Circular Queue
  - Properties of a Circular Queue
  - Operations of Circular Queue
  - Applications of Circular Queue

Limitation of Linear Queue

The only limitation of a linear queue is that- If the last position of the queue is occupied, it is not possible to enqueue anymore elements even though some positions are vacant.

Operation
Enqueue(g)

Rear	front
5	3

0	1	2	3	4	5
			d	e	f

Error: Queue is overflow

- **Solution: Circular Queue**
  - This limitation can be overcome by moving the rear back to index '0', if front is >0

Circular Queue

- Circular Queue is the advanced form of Queue data structure.
- Like a linear Queue, elements are added from an end i.e. **rear**, and removed from another end that is known as the **front**.
- It also ensures the first-in-first-out (FIFO) or last-in-last-out (LILO) order of insertion and deletion.
- However, Unlike linear queue, in circular queue rear is reset to index '0', if there are some vacant slots at the beginning.

5

Linear Queue vs. Circular Queue

Linear Queue

Operation	Rear	front
Enqueue(g)	5	3

0	1	2	3	4	5
			d	e	f

Error: Queue is overflow

Circular Queue

Operation	Rear	front
Enqueue(g)	0	3

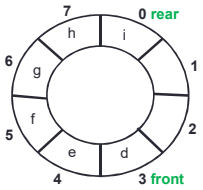
0	1	2	3	4	5
g			d	e	f

6

Circular Queue

Operation	Rear	front
Enqueue(i)	0	3

0	1	2	3	4	5	6	7
i			d	e	f	g	h



7

Operations of Queue

- The common operations of queue are as follow:
  - enqueue()
  - dequeue()
  - isEmpty()
  - isFull()
  - frontValue()
  - rearValue()

8

Operations of Queue-Enqueue(item)

Enqueue (queue, item)

1. If queue is already full: //front=0 & rear=size-1 or rear+1=front
2. Display an error of "overflow"
3. If queue is empty and this is the first item to be inserted in that queue //rear==front==-1
4. Increment rear and front both
5. Insert item at rear index
6. If queue is not empty and there are some vacant slots in the begining: // front > 0 and rear==size-1
7. Set rear=0
8. Insert item at rear index
9. Else: there are some slots after rear
10. Increment rear
11. Insert item at rear index

9

QUEUE- Enqueue Operation

Operation	Rear	front	0	1	2	3	4	5
-	-1	-1						
Enqueue(a)	~~-1~~ 0	~~-1~~ 0	a					

Increment rear and front both and then insert item at rear

Enqueue(g)	5	0	a	b	c	d	e	f

Overflow because front is at 0 and rear is at size-1

Enqueue(e)	~~3~~ 4	2			c	d	e	

Increment rear and then Insert item at rear index

Enqueue(g)	~~5~~ 0	2	g		c	d	e	f

If queue is not empty (front > 0 and rear==size-1) then set rear=0 and Insert item at rear index

10

Operations of Queue-Dequeue()

Dequeue (queue):

1. If Queue is already empty: // front==rear==-1
2. Display an error of "underflow"
3. If there is only one element in the queue //front==rear && front!=-1
4. Save value of front index in a variable Data
5. Set front and rear both to -1
6. Return Data
7. If there are more than one elements with conditions front==size-1 &&front>rear
8. Save value of front index in a variable Data
9. set front=0
10. Return Data
11. Else: there are some elements after front
12. Save value of front index in a variable Data
13. Increment front
14. Return Data

11

QUEUE- Dequeue Operation

Operation	Rear	front	0	1	2	3	4	5
-	-1	-1						

underflow

Dequeue()	~~-2~~ -1	~~-2~~ -1			~~e~~			

Save front value as Data, set rear and front both at -1and then return Data

Dequeue()	5	~~0~~ 1	~~a~~	b	c	d	e	f

front < rear: Save front value as Data, increment front and then return Data

Dequeue()	4	~~2~~ 3			~~e~~	d	e	

front < rear: Save front value as Data, increment front and then return Data

Dequeue()	1	~~5~~ 0	g	b				~~f~~

front == size-1: Save front value as Data, set front=0 and then return Data

12

Enqueue - Dequeue Operations

Operation	Rear	front	0	1	2	3	4	5
Dequeue()	5	~~1~~ 2		~~b~~	c	d	e	f
Dequeue()	5	~~-2~~ 3			~~e~~	d	e	f
Enqueue(g)	~~5~~ 0	3	g			d	e	f
Dequeue()	0	~~3~~ 4	g			~~d~~	e	f
Enqueue(b)	~~0~~ 1	4	g	b			e	f
Enqueue(j)	~~1~~ 2	4	g	b	j		e	f
Enqueue(k)	~~2~~ 3	4	g	b	j	k	e	f
Dequeue()	3	~~4~~ 5	g	b	j	k	~~e~~	f

13

Total Enqueue	Total Dequeue	Rear	front	0	1	2	3	4	5
							DD	EE	FF
				GG	HH	II			MM
				AA					FF
					MM	NN			

14

Operations of Queue-isFull()

isFull():

1. If front=0 && rear==size-1 or rear+1==front:
2. Return true
3. Otherwise:
4. Return false

Operation	Rear	front	0	1	2	3	4	5
isFull()-True	5	0	a	b	c	d	e	f
isFull()-False	5	3				d	e	f
isFull()- False	3	0	a	b	c	d		
isFull()- True	3	4	g	h	i	j	e	f

15

Operations of Queue-isFull()

isEmpty():

1. If rear and front are at -1:
2. Return true
3. Otherwise:
4. Return false

Operation	Rear	front	0	1	2	3	4	5
isEmpty()-True	-1	-1						
isEmpty()-False	5	3				d	e	f
isEmpty()-False	3	0	a	b	c	d		
isEmpty()-False	3	4	g	h	i	j	e	f

16

Operations of Queue-frontValue()

frontValue():

1. If rear and front are at -1:
2. Display error “underflow”
3. Otherwise:
4. Return value at front index

Operation	Rear	front	0	1	2	3	4	5
frontValue() Error	-1	-1						
frontValue() e	3	4	g	h	i	j	e	f

17

Operations of Queue-rearValue()

rearValue():

1. If rear and front are at -1:
2. Display error “underflow”
3. Otherwise:
4. Return value at rear index

Operation
rearValue() **Error**

Rear	front
-1	-1

0	1	2	3	4	5

Operation
rearValue()
j

Rear	front
3	4

0	1	2	3	4	5
g	h	i	j	e	f

18

Applications of Queue

- It is used in all those application where FIFO/LILO order is mandatory.
- It is used for scheduling purpose
- It can be used for buffering of data packets, where order of packets must be maintained

19

Thank You