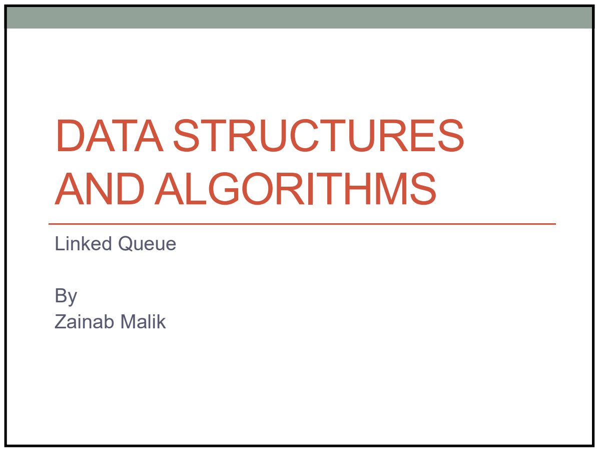
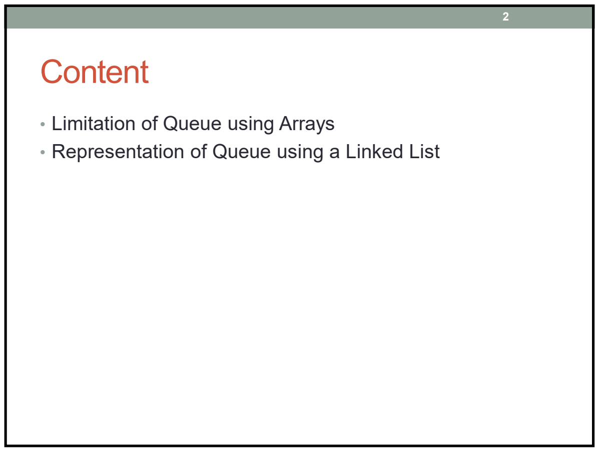


DATA STRUCTURES AND ALGORITHMS

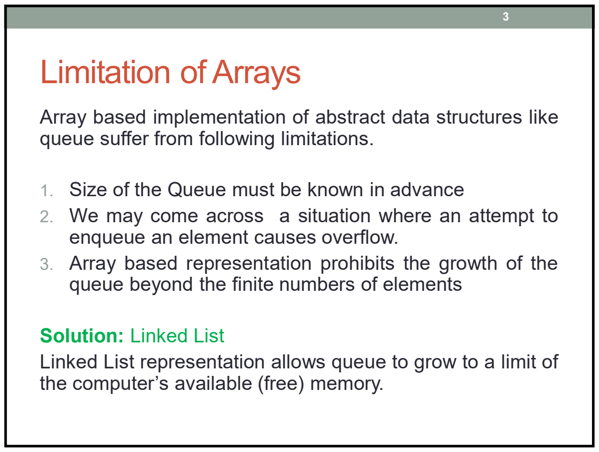
Linked Queue

By
Zainab Malik



Content

- Limitation of Queue using Arrays
- Representation of Queue using a Linked List

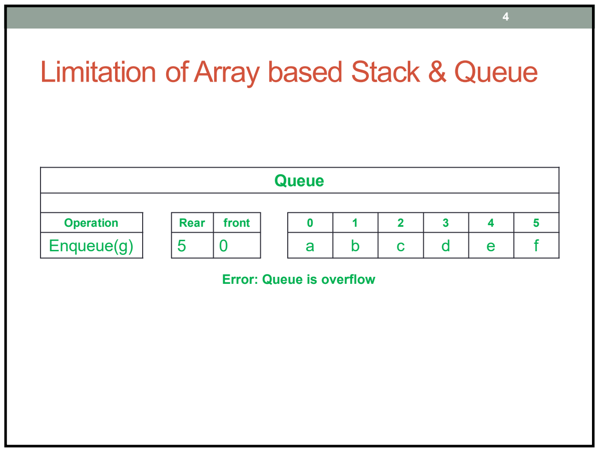


Limitation of Arrays

Array based implementation of abstract data structures like queue suffer from following limitations.

1. Size of the Queue must be known in advance
2. We may come across a situation where an attempt to enqueue an element causes overflow.
3. Array based representation prohibits the growth of the queue beyond the finite numbers of elements

Solution: Linked List
Linked List representation allows queue to grow to a limit of the computer's available (free) memory.



Limitation of Array based Stack & Queue

Queue

Operation	Rear	front	0	1	2	3	4	5
Enqueue(g)	5	0	a	b	c	d	e	f

Error: Queue is overflow

Solution:

Linked List Representation of Queue
or
Linked Queue

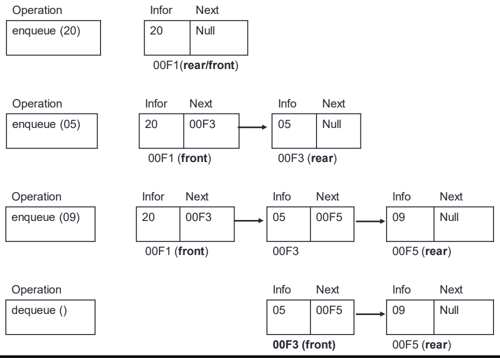
Linked List Representation of Queue

- Just like a singular linked list, in linked queue, each node has two slots
  - First slot contains the information/content
  - Second slot contains the pointer/address of the next node

Info	Next
12	Null

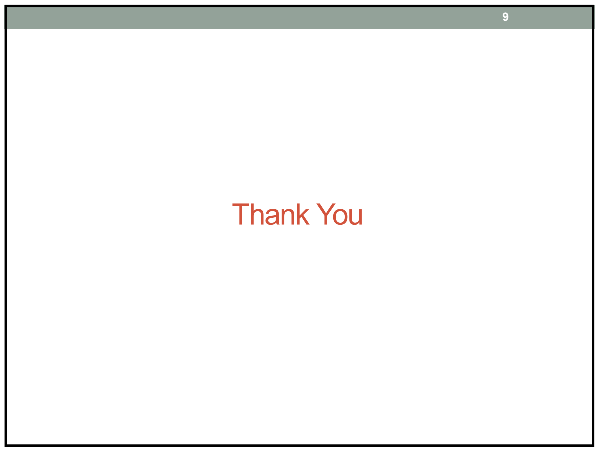
- Instead of head and tail pointers, in a linked queue there are two pointers “front” and “rear”
  - Front pointer holds the address of first node (like head pointer)
  - Rear pointer holds the address of last node (like tail pointer)
  - When there is no element in the linked queue, both pointers contain null
  - To ensure FIFO or LILO order
    - The enqueue operation is implemented by inserting a new node at the end of the list (**AddAtEnd()**)
    - The dequeue operation is implemented by removing the node from the beginning of the list (**RemoveFromStart()**)

Linked List Representation of queue



Operation of Linked Queue

- **enqueue(item):** We may implement function same as addAtEnd() function of singular linkedlist (AI-Lecture03)
- **dequeue():** We may implement function same as removeFromBeginning() function of singular linked list (AI-Lecture03)
- **isEmpty():** Need to check front and rear pointers, if both contain null or 0 it means that linked queue is empty
- **isFull():** No need to implement because there is no fixed size in linked list representation of queue
- **rearValue():** Need to return rear->info
- **frontValue():** Need to return front->info
- **removeAll():** this is also known as destructor in which we delete all nodes one by one till front becomes 0
  - Which function to call for the deletion of these nodes?



Thank You