

This lecture we will learn about

1. Linked Implementation of Queue
2. Definition of Deque
3. Implement Deque using Arrays
4. Implement Deque using Doubly Linked List.

Linked List implementation of Queue

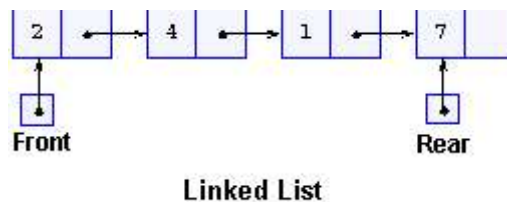
In the previous lecture, we implemented queues using arrays. This lecture we are going to implement queue using linked list, like we implemented stacks using linked list.

Drawback of arrays are

1. Arrays lead to wastage of memory – Because size of array is fixed, if number of elements happen to be less than the size of array, then rest of the left over array is not utilized.
2. Arrays are static – Because size of array is fixed, if user want to enter more elements than the size of array, he can't do so, he will confront queue full condition.

Linked list are dynamic and hence overcome all the drawbacks of arrays i.e. There is no wastage of memory.

A linked list where we can insert after the last node and can delete only the first node is linked implementation of queue.



The above figure is a linked implementation of queue, If we want to insert we can only insert after the last node i.e. We can only insert after the rear. If we want to delete, we can only delete the first node, i.e. We can only delete the node pointed by front.

Or We can say, **In linked implementation of queue, front points to the first node and rear points to the last node.**

Definition of Deque

Deque is an acronym for double ended queue.

We have learnt that linear queue has 2 ends, front and rear. In linear queue, insertions are done from rear and deletions are done from front.

Double ended queue is a data structure in which insertions and deletion can be done from both the ends, i.e. We can perform insert and delete from the front, also we can perform insert and delete from the rear. Therefore we defined 4 operations with deque

1. Insert from Front – `insert_front()`
2. Delete from Front – `delete_front()`
3. Insert from Rear – `insert_rear()`

4. Delete from Rear – `delete_rear()`

To understand how a deque works, watch the presentation.

Below is a summary of presentation:

1. A deque is empty, if condition $(\text{front} == -1 \text{ and } \text{rear} == -1)$ is true
2. A deque is full, if condition $(\text{front} == 0 \ \&\& \ \text{rear} == \text{size_of_array} - 1)$ is true.
3. A deque has only one element, if condition $(\text{front} == \text{rear})$ is true.
4. For operation Insert from Front,
 - 4.1. Check if deque is full $(\text{front} == 0 \ \&\& \ \text{rear} == \text{size_of_array} - 1)$, if deque is full display message "Cannot insert, deque is full".
 - 4.2. If deque is not full, Check if we are inserting for the 1st time i.e. Condition $(\text{front} == -1 \ \&\& \ \text{rear} == -1)$ is true. If condition is true, then set $\text{front}=0$ and $\text{rear}=0$, and insert data at position front.
 - 4.3. If deque already contains more than 1 element, then check condition $(\text{front} != 0)$ is true. If condition is false, then decrement front and insert data at position front.
 - 4.4. If condition $(\text{front} != 0)$ is false, this means front is equal to 0, then shift all the elements from position front until rear ahead by 1 position. Insert data at position front.
4. For operation Insert from Rear,
 - 4.1. Check if deque is full, condition $(\text{front} == 0 \ \&\& \ \text{rear} == \text{size_of_array} - 1)$ is true, if deque is full, display message "cannot insert, deque is full".
 - 4.2. If deque is not full, Check if we are inserting for the 1st time, i.e. Condition $(\text{front} == -1 \ \&\& \ \text{rear} == -1)$ is true. If condition is true, then set $\text{front}=0$ and $\text{rear}=0$, and insert data at position front.
 - 4.3. If deque already contains more than 1 element, then check condition $(\text{rear} != \text{size_of_array} - 1)$ is true. If condition is true, then increment rear and insert data at position rear.
 - 4.4. If condition $(\text{rear} != \text{size_of_array} - 1)$ is false, that means rear has reached the end of array. Shift all the elements from position front until rear back by 1 position. Insert data at position rear.
5. For operation Delete from Front,
 - 5.1. Check if deque is empty, if $(\text{front} == -1 \ \&\& \ \text{rear} == -1)$ is true, this means deque is empty. Display message "Cannot delete because deque is empty".
 - 5.2. If deque is not empty, check if we are deleting the last element. If condition $(\text{front} == \text{rear})$ is true, this means there is only 1 element left in the queue. In this case, get the value of element from position front and set $\text{front} = -1$ and $\text{rear} = -1$.
 - 5.3. If there are more than 1 element in the deque, i.e. Condition $(\text{front} == \text{rear})$ is false, then get the value from position front. Increment front by 1.
6. For operation Delete from Rear,
 - 6.1. Check if deque is empty, if $(\text{front} == -1 \ \&\& \ \text{rear} == -1)$ is true, this means deque is empty. Display message "Cannot delete because deque is empty".
 - 6.2. If deque is not empty, check if we are deleting the last element. If condition $(\text{front} == \text{rear})$ is true, this means there is only 1 element left in the queue. In this case, get the value of element from position front and set $\text{front} = -1$ and $\text{rear} = -1$.
 - 6.3. If there are more than 1 element in the deque, i.e. Condition $(\text{front} == \text{rear})$ is false, then get the value from position front. Decrement rear by 1.