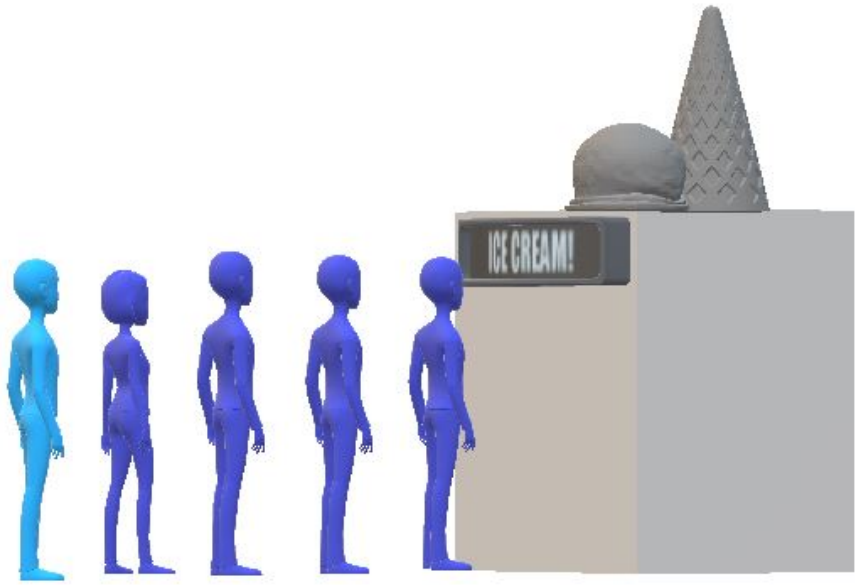
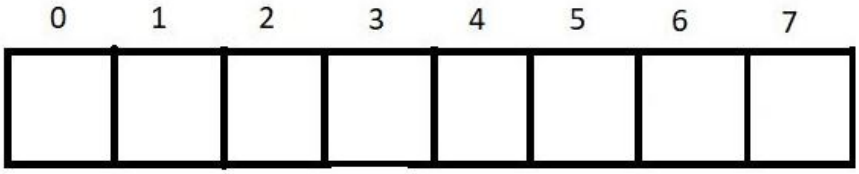
Queue Implementation: Array Implementation of Queue in Data Structure

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-39

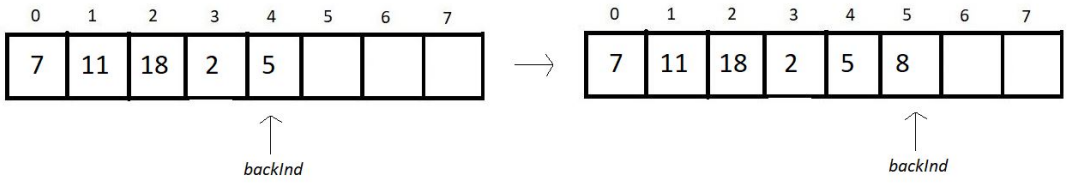
In the last lecture, we introduced to you a new data structure, queue. Today, we'll learn how to implement queues ADT using arrays. During our discussion, we compared its representation to our own lives. It is analogous to a queue in front of any ticket counter or an ice cream shop illustrated below.



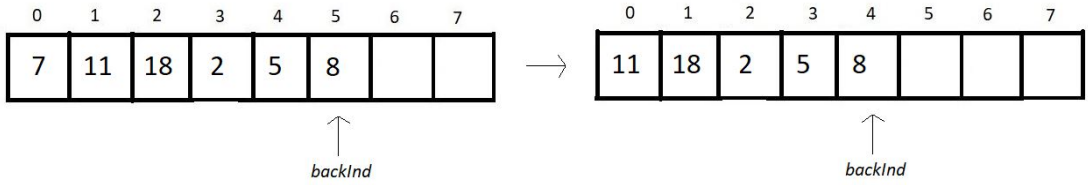
Here, we have shown a branded ice cream shop that is famous enough to have a queue of people waiting to get one of their choices. And the shop owner wants to store the information of these people, so he uses an array to accomplish that. Assuming that we have 8 people and we want to store their information, we'll have an array as illustrated below:



Here, we'll maintain an index variable, *backInd,* to store the index of the rearmost element. So, when we insert an element, we just increment the value of the *backInd* and insert the element at the current *backInd* value. Follow the array below to know how inserting works:



Now suppose we want to remove an element from the queue. And since a queue follows the FIFO discipline, we can only remove the element at the zeroth index, as that is the element inserted first in the queue. So, now we will remove the element at the zeroth index and shift all the elements to its adjacent left. Follow the illustrations below:



But this removal of the zeroth element and shifting of other elements to their immediate left features O(n) time complexity.

Summing up this method of enqueue and dequeue, we can say:

1. Insertion(enqueue):

- Increment *backInd* by 1.
- Insert the element
- Time complexity: O(1)

2. Deletion(dequeue):

- Remove the element at the zeroth index
- Shift all other elements to their immediate left.
- Decrement *backInd* by 1

3. Here, our first element is at index 0, and the rearmost element is at index *backInd*.

4. Condition for queue empty: *backInd = -1*.

5. Condition for queue full: *backInd = size-1*.

Can there be a better way to accomplish these tasks? The answer is yes.

We can use another index variable called *frontInd,* which stores the index of the cell just before the first element. We'll maintain both these indices to bring about all our operations. Let's now enlist the changes we'll see after we introduce this new variable:

1. Insertion(enqueue):

- Increment *backInd* by 1.
- Insert the element
- Time complexity: O(1)

2. Deletion(dequeue):

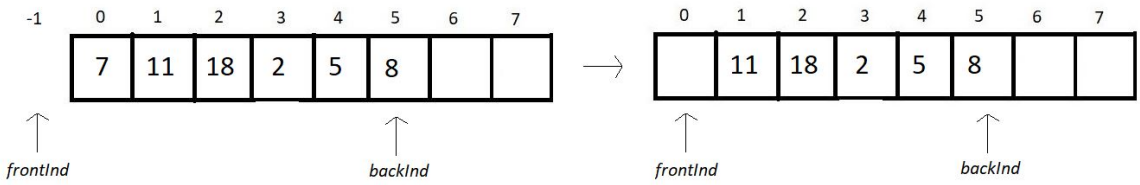
- Remove the element at the zeroth index(no need for that in an array)
- Increment *frontInd* by 1.
- Time complexity: O(1)

3. Our first element is at index *frontInd* +1, and the rearmost element is at index *backInd*.

4. Condition for queue empty: *frontInd = backInd*.

5. Condition for queue full: *backInd = size-1*.

Now, we were able to achieve both operations in constant run time. And the new dequeue operation goes as follow:



The act of optimizing a solution/program is very important, and you should always strive for a better solution to a problem. And a solution that takes less time is always preferred. So, this is how we implement the queue ADT using an array. The programming part is still left, which will be covered in the next tutorial. But before you proceed, make sure you are finished with all the fundamentals, and if not, watch the video and go through the lectures again.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube

channel to access it. See you all in the next tutorial, where we'll learn programming to implement the queue ADT using arrays. Till then, keep coding.