www.google.com redirected you too many times.

Show Course Contents

Overview Q&A Downloads Announcements

Array implementation of Queue and its Operations in Data Structure

In the last tutorial, we discussed the idea of the implementation of queues using arrays. We talked about the basic operations and their best methods. And we came to the conclusion that if we maintain two index variables, *frontInd & backInd*, we can accomplish both enqueue(insertion) and dequeue(deletion) in constant time complexity. Let me just enlist the method we prepared:

1. Insertion(enqueue):

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

2. Deletion(dequeue):

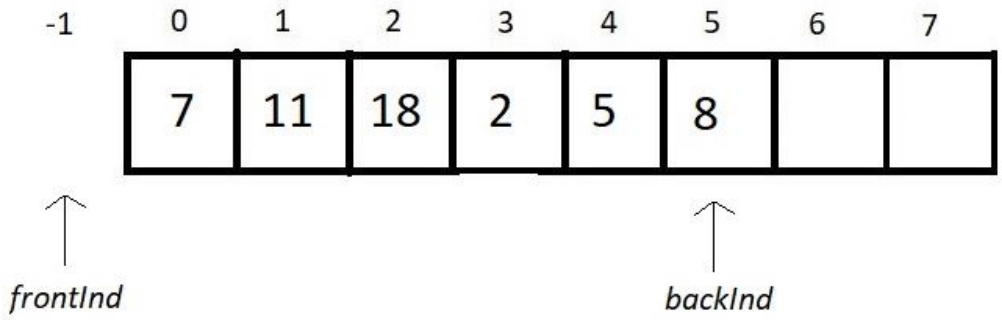
- Remove the element at the zeroth index(no need for that in an array)
- Increment *frontInd* by 1.
- Time complexity: O(1)

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.*

Given array below represents a queue:



To implement this, we'll use a structure and have the following members inside it:

1. size: to store the size of the array

2. frontInd: to store the index prior to the first element.

3. backInd: to store the index of the rearmost element.

4. *arr: to store the address of the array dynamically allocated in heap.

```
struct queue
{
    int size;
    int frontInd;
    int backInd;
    int* arr;
};
```

Now to use this struct element as a queue, you just need to initialize its instances as:

1. struct Queue q; (we are not dynamically allocating q here for now, as we did in stacks).

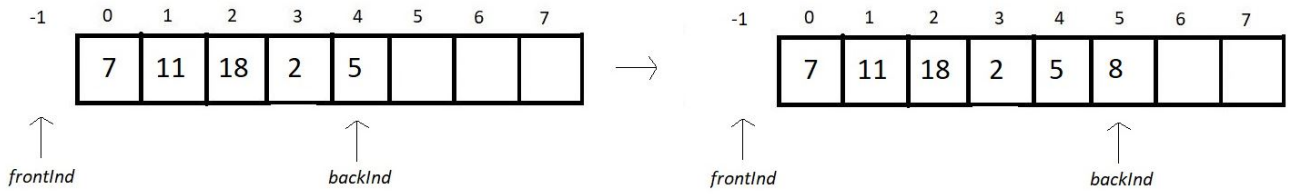
2. Use dot here, and not arrow operator to assign values to struct members, since q is not a pointer.
3. q.size = 10; (this gives size element the value 10)
4. q.frontInd = q.backInd = -1;(this gives both the indices element the value -1)
5. Use malloc to assign memory to the arr element of struct q.

And this is how you initialize a queue. We will now devote our attention to two important operations in a queue: enqueue and dequeue.

Enqueue:

Enqueuing is inserting a new element in a queue. Prior to inserting an element into a queue, we need to take note of a few points.

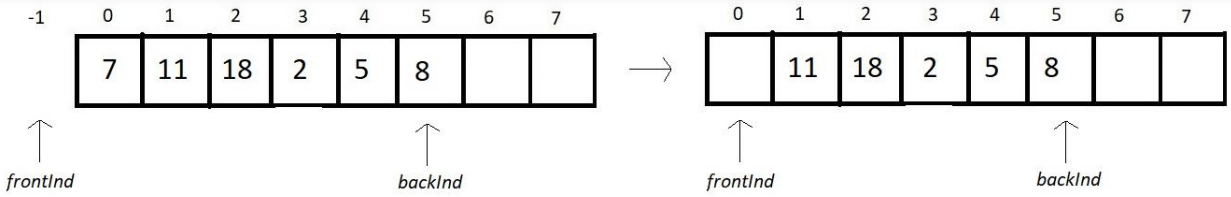
1. First, check if the queue is already not full.
2. If it is, it is the case of queue overflow, else just increment *backInd* by 1, insert the new element there. Follow the illustration below.



Dequeue:

Dequeuing is deleting the element in a queue which is the first among all the elements to get inserted. Prior to deleting that element from a queue, we need to follow the below-mentioned points:

3. First, check if the queue is already not empty.
4. If it is, it is the case of queue underflow, else just increment *frontInd* by 1. In arrays, we don't delete elements; we just stop referring to the element. Follow the illustration below.



Condition for *isEmpty*:

1. If our *frontInd* equals *backInd,* then there is no element in our queue, and this is the case of an empty queue.

Condition for *isFull*:

1. If our *backInd* equals (the size of the array) -1, then there is no space left in our queue, and this is the case of a full queue.

So, these are the basic operations of a queue. We have implemented everything using arrays. We'll see the programming part in the next tutorial. Keep up with the playlist, and you won't miss anything.

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 and its operations using arrays. Till then, keep coding.

Previous Next

CodeWithHarry