Data Structure

Lecture 5

Definition of Queue:

Queue is a very important data structure and it is a linear data structure. This is a typical example of a queue. So as you can see - a person who is standing first in the queue which is called the front of the queue technically, who is the first person to come out and a person who wants to join the queue or in other words adding on to the queue is done at the rear of the queue- the back side of the queue. This is deletion- deletion is done at the front and the addition is from the back.

Queues in our life, it is not that we have seen for the first time, we have seen queues in our everyday life like when you stand in a queue to buy a bus ticket or when you stand in a queue for visiting a temple and so on…so the same concept is what you are going to in computer science also where queue is actually a list of objects or list of items that are waiting to be added or waiting to be deleted.

This is what called the front of the queue and this is called the rear of the queue - in another words the front acts as the beginning of the queue and the rear acts as the ending of the queue. And we call this…that is if you want to delete an element we call it as remove or in terms of queue it is called technically Dequeue an element and it is always done from the front and the addition of the element is done at the back which is called Insertion or Enqueue technically. So, this is a queue when we talk in terms of computer science. This is a queue where we have normally... in a bank, in a temple and so on….

A queue is similar to a list - now we have seen lists, we have seen stacks. So this is... Queue is similar to a list but adds items - it is a constraint that we put on where we can add or where we can delete - that queues are different. So Queue adds item only at the end of the list and removes them from the front. So this is called the First in and First out Structure or a policy- Whoever comes in first will leave the queue first this is also true for everyday queues. The first element inserted will be the first one to be removed. Now you may think that in certain temples like Thirupathy and all that we have what are called as priority queues - meaning that it is not always first in first out. So in those cases - even in computer science we have cases where you have to meet deadlines where certain objects should be processed first - in such cases you can add elements in the middle of the queue according to some priority order.

How do you work with Queues?

We have seen enough of queues in our everyday life to know the conceptual idea of a queue and we have to have a set of operations that we need to process the

queue and actually we do not care about how it is exactly implemented and this is conceptual idea.. This is a black box of implementation so before we go into the actual implementation from the Computer science view point - let us look at all the operations. So let us continue with the list of elements. Normally a queue will consists of a list or a group of homogeneous elements normally it is saying like that you can't have a queue of animals and people standing together they'll all be people. Like that in computer science you'll normally have certain items that have been defined - all of them are homogeneous items and of course it is a first in first out always and this you'll have to remember always - it is called as FIFO or First in First out. So when you think of a queue you should immediately think of FIFO policy. As already described elements can be added only at one end - now we are bring in the technical term - the Rear of the Queue and remove only from the end of the queue called the Front of the Queue. So the length of the queue is the number of items it contains. An empty queue will obviously have a length of zero. So these are an analogy in the bank's teller's window so we will enqueue here and dequeue here. This dequeue is deletion and enqueue is insertion. And as we have already said these are queue elements – any one of these elements - you can define it, it can be a structure, it can be an integer, it can be a name, it can be a string or it can be a real number. So the elements can be any one of these, only thing is that it has to be homogeneous and this is the rear of the queue and this is the front of the queue. Normally you say the queue length is four and you can also have an empty queue. So just to compare with stacks.
Stacks follow the Last in First out that is whoever entered the stack last will be deleted from the stack first so that is called "Last inside First outside" (LIFO). On the other hand queue is 'Fist in First Out'. So in stack you push into the one end of the stack and pop out from the same end of the stack so whoever has gone into the stack last will come out first. But in queue it is other way round whoever comes first will be deleted and whoever comes in the end will be put in the back. Let's look at the typical conceptual example of an animal parade. Here you add called as enqueue and you have a front and in the front you have added Ant and this is the back. Then in this queue - you are enqueueing again another animal Bee at the back (rear) (please note that this is added at the back of Ant) then you have Cat then you have Dog and so on.. So this is a typical example of an animal parade queue. Now let's look at it from another view point so now you have A (please note that the while in a stack the access point is only one end - in a queue you have two access points the front and the rear) in front you insert and in the back (rear) you delete. So this is an example of a queue which has been created - so here A has been removed.. Here first you have entered A first then you have entered B then you have entered C then you have entered D. Suppose you want

to delete or dequeue, you can't delete D which we did in a Stack, we need to delete A which has come first as that is First In and First Out.

You may think of where all Queues can be used. There are many application of queues, in computer science even and you can have scheduling jobs/processes in the computer and the giving housing policy or something like that, computer simulations you have the queue where you know, whoever has first come in to the system should be processed first and queues are often helpful in simulations and in any processing in which items get backed up. That is you have too many items and you have a rare resource that you have to share among these items then normally without any prejudice what you do is whoever is coming first will be processed first so that is called the backed up Processes.

Queue Implementations

When you have a conceptual model of a queue obviously in a computer science course you have to know how to implement the queue. So you can use a vector, linked list or any sequential structure. Linked list is an easy method or a better method of implementing stack and queue because you have to constantly in applications add and remove nodes. The vector representation that is an array representation of a queue is tricky - we will go into the application later. Now we'll see what the problem in an array implementation of a queue is? The trick here is that there is a difference in Full and Empty and I will come into that later as we go along. So as I have already told you - backed up I told you in a non-technical terms - in other words in computer science terminology it is called as buffering so the purpose of a queue is to provide some type of buffering. Typical use of queue in computer systems are process management where you have too many processes sharing a single CPU and you want the processes to be given time for processing or - CPU time then you do it according to queue policy or when you have a fast computer and a slow printer so what you do is.. you have too many items to print on the printer so you put them in a queue so it is called the print queue and one by one the items are taken out and then printed on the printer so here what happens is the first item or the job that is put in front is processed or printed first

Operations on Queues:
These operations are similar to most data structures when you take list or stack you have these operations. The only thing is that how these operations are defined is slightly different. So Make Empty is to make a queue empty and the Length (Q) is to obtain the length of the queue.
Access function is –these are all access functions - when you do not do any

modification to the queue – that the queue remains as it is but you want to access it. So this is an Is Empty - this is to test whether the queue has elements or not - for example you are doing a printing operation and then it is queued and then you want to switch off the computer - you want to know whether there are any jobs still awaiting printing and this is just to test whether the queue is empty. This is to know or to access the first element - the front element of the queue. Front (Q) is used to just inspect it – u do not remove it or anything - to know who is first in the queue may be and if it is empty then you cannot access it. Then End (Q) is to see who is last - who would be the last to be processed and again you have to check for the empty condition. And as you see these three - the queue initially - whatever the queue is the queue remains as it is, only thing you are accessing, one is the Boolean function it is true if it is empty and false if it is not empty, this gives the front element of the queue and this gives the last element or the rear element of the queue.

Then you have manipulation procedures. This we have already seen enqueue, so enqueue means insertion or addition you can call it whatever you want but in Queue terms it is normally called as enqueue but some materials talk about it as insertion and some talk about it as addition but all mean the same thing. So you add an element to the end of the queue. Please note here you have no other option you have to add it in the end only. So this is the queue so this is to access the queue that has been implemented - the queue structure and this is the element that you want to insert. So all insertions are done at the back so the function of this enqueue is to add an element at the rear end - so ads X to the rear of the queue. Precondition that you have to check is that it is has been initialized - the queue must be available and it should not be full because for every addition operation you have to check whether you can add first of all - because it should not be full. The post condition is that the new item must be put at the back of the queue and obviously the length of the queue would have been increased by one. But, suppose by this you have reached an overflow that is you cannot add any more items - may be you can also signal the overflow condition here. Now this depends on the implementation each implementation will be slightly different.

Dequeue - as we have already said this is removing an element or deletion of an element. In most material it is called as Dequeue - so dequeue means deleting an element from the queue so as I have told you, you can only delete from the front - there is no other option. So to delete from the first – all deletions are done from the front. Now here the function is to remove the front element here you don’t have to say which element has to delete because we are going to delete only the back element so, the function removes. The precondition is obviously it has to be initialized and it should not be empty and it should contain at least one element. The post condition is that the front element has been removed from the queue and the copy of the element is available for access and the length of the queue has been reduced by one and but if it is empty we have to check whether it has become empty now - then you have to signal the empty condition. So these are the basic operations that we do with queues.

What are Queues?

Computer science we normally use queues- one of the uses of queues is to store values. The values can represent jobs; can represent processes and whatever it might be. So this is a typical queue of integers where we have 4, 3,10,7 as the integers and then it is a queue of length four as you can see and then now we want to insert an element - remember the enqueue operation was - you have to access this queue and X is the element that you want to insert - so 8 is the element you want to insert. After the enqueue operation - please note that after the enqueue operation - this is the front of the queue and this is the rear of the queue. The addition operation is always done at the back so this is the modified queue. Similarly you have a queue of integers the length of the queue has increased by one. Here the length of the queue is five. Now we only delete from the front and here we are going to delete 15 - there is no question of deleting any other element. So you dequeue from the front and the length of the queue has decreased by one, so this modified queue.

The enqueue operation - so this is without the element - so any element can be added here. Now let us take an example and see how this is done. Now let us assume that initially the queue has two elements apple and mango and this is a queue and after the enqueue operation you have the data and you have the Apple and queue. You have element that has been added so this is the element that you want to add and this is the operation you do and this is the result of the operation. So you want to add orange say X, remember enquue and this is a queue and enqueue(X), and here X is orange. The result of the operation is the orange has been added to the rear. The important point to note here is that this is the only place we can add - nothing else can be done.

The dequeue. This element is returned back by the queue. So the dequeue deletes and returns the element. So this is the initial queue so you can only delete from the front so what happens is when we dequeue element gets deleted and the front element is deleted and the length of the queue is decreased by one. So now let’s look at dequeue so what happens is you have an element an Apple, Mango and Orange and you have to delete from the front of the queue - Apple and there is no question of specifying the element because you can only delete the front element. Let us take the operation- the dequeue operation - now after the dequeuing operation now this apple is accessible – it is now taken out and the result is this queue – which has apple removed from it. So let us take another example the

Queue Front - please note that the queue before and after the Queue Front operation remains the same - only thing you can access what you can access is the front of the element. So when you do Queue Front this element Apple becomes accessible that is all. Then similar for rear - Queue Rear and here you have Queue Rear and what is accessible is the rear element which is orange.
Let us take an example of a queue which is an empty to start with and you are trying to enter colors into the queue. So what happens - is you have an empty queue where you enqueue Green and then you want to enqueue Blue so and then blue also gets enqueued and then you want to dequeue green so what is at the front is removed. Then you want to enqueue red and so red is added and then you want to just access blue - the front element now that has been accessed, you want to access the last element the red that has been accessed and then you want to dequeue so what is dequeue is the front element which is blue and then you want to dequeue Red which is the front & the rear element - only one element so you have removed it - so what you have it is Queue empty.

Implementation of Queue as Arrays

This is a typical array implementation of queues. So you have 0 to 16 as the array elements and what happens is that you can just start with 0. Here we are assuming that the array starts from 5 and goes on to 11 so this is the queue front (5) and this is queue rear (11). Now what happens sometimes if you keep on doing queue front and queue rear - this something that we have explain very carefully. What happens is if we keep on removing elements this will be keep on moving when you also add but in spite of that there are 4 or 5 locations here which are free but still queue rear will still indicate an overflow condition because, you normally implement the array as starting from 0 to 16 so when the rear becomes greater than 16 you will attain an overflow condition. But however please note since you have implemented the queue from the number of elements that have been removed so the queue is actually not full as there are 5 extra spaces available here but because the queue rear is greater than 16 you would have signalled a queue full condition. So this means that even if the queue does not contain 17 elements that is 0 – 16 is 17 elements you are signalling an overflow condition which is not right. It means that you not utilizing the array in a proper way. Now this a problem with linear arrays. Now if you look even you could remove all the elements in the array and front would have been to pointing 15 even then - the next time you have an enqueue condition - you still would get a queue full condition which does not make sense isn’t it. This is not the way in which you want to implement so what you do - you keep on increasing the size of the array - that is not a solution.

As you saw here – this is a queue array – the maximum size is seven. So you keep on implementing – just to show this in a specific form. So this is the array implementation – now this is the queue front and this is the queue rear. Now what we are going to do to solve of this problem is – we are going to bend this such that it forms a circular queue. Now look at this -we are bending this to form a circular queue. This is what we are doing. Now this is just our imagination – physically it is still a linear array – but now we are conceptualizing the linear array as a circular array. This is one of the main concepts in queues which you have to understand where an array when it is used for queue implementation should be treated like a circular array. Now look here – this is the queue front – this is the queue rear. Now when will you get a queue full condition? Only when all the elements have been filled. Let us assume that this is the first array element and this is the last array element and this in our previous example this could be zero and this could be some 30 or 25 or 35 how many ever locations you want. So the front will not be necessarily the first element of the array and the last need not be necessarily the last so it can be in between too so at both the conditions so you will not get a queue empty or a queueful condition unless the queue is actually full. But here also there is a small problem. How do you find out whether the queue is actually empty or the queue is full? Now suppose I start my queue from here and it goes on growing and up to this I fill it okay? What will happen? I'll get a queueful condition that I think you can imagine. But, what happens is suppose it is queue empty - this will be still next to each other but all this will be empty which means there is no difference between the queue full and the queue empty condition. So one solution to this is to leave one empty space when the queue is full so you'll leave one empty space when the queue is full and that will tell you that this is when the queue is empty the front and the rear can point to the same location. When the queue is full you go only up to (rear -1) - don't go up to rear. So what will happen it becomes an empty space and so the front and the rear will not point to the same location. So this way you can differentiate between empty and full condition and you can utilize the queue completely. So in the linear array, the number of elements you use when queue full will be signaled can vary from 0 or 1 up to the maximum size of the array or the maximum number of elements that can be accommodated. But in circular queue you'll get a queue full condition only when all the locations are occupied except one location which we has chosen to leave as empty so that you can differentiate between the queue full and the queue empty condition . So this basically the implementations of queues using what we call as circular arrays. So we have seen how queues can be implemented using arrays but normally we do not use the linear representation of arrays we do try to bend it forming conceptually circular queue and then do the implementation such that I am utilizing the locations of the array completely or at least one less when I signal the queue full condition. This is one of the main concepts that we have to see in queues. Now we have to look at the other implementation of queues which is the linked list representation of queues.

How to use linked list to Implement Queues

How to use linked list to implement queues? This is similar to the list implementation. Before we go into the details - remember that queue implementation and stack implementation - all of these are basically constrained list – list is a general purpose data structure. Stack is a data structure which is again a list which follows the Last In First Out policy while queue is a data structure that follows the First In First Out policy so it is equivalent to saying that I am going to do insertion only at the back of the list and deletion is only from the front of the list in case of a queue. In case of a stack the insertion is done at only one side of the list or in top may be and the deletion also is from the front of the stack so that is how we were doing.

Now similar, to that we can talk about the implementation of queues. This is the front and rear to access the queue. Remember we need two access points to the queue – the front and rear. Then each cell of the queue will have the value it can be an integer, it can be a string or whatever where you have a portion which gives address of the next element - the logical address of the next element.