Data Structure

Lecture 6

Implementation of Queues

In queues we are going to discuss the implementation of queues using linked list, what is meant by priority queues and we are going to look at a typical application of queues that is the breath first search. So, first let us talk about the implementation of queues. So we are going to look at how the linked list is used to implement queues. So for this we have to declare two variables called front and rear with which we are going to access the queue. As we have already seen in the last lecture the front is used for deletion purposes and the rear is used for insertion. So the node of the queue consists of a value along with a pointer to the next node. So looking at this we have a queue has the elements 4, 3, 7 and 8 now it is put in a node of a linked list with the data being here and this being the address of the next element. Now this is a typical list that we have already seen. However you can access the list by two methods - one is through the front of the list and another is through the rear of the list. The front is used for deletion purposes and the rear is used for insertion purposes so, this is an object class queue - this is the pointer to the queue so the queue is actually accessed only through the two pointers that is the front and the rear. Here you can see that this is a queue with a single element 9 and as you can see - the front and the rear both point to this element so if you want to insert an element you'll insert it here or if you want to delete an element - delete this element and the front becomes 0. This is an indication that there is no next element this is the only element in the queue which means that you have only a single element queue. So you can also have a storage for a queue as a queue array with a maximum size and you can have a count of the number of elements in the queue but again you have to have two pointers or two access points which are the front and the rear where front is used for deletion purposes and rear is used for insertion purposes.

Create a queue: So before we start off with the implementation of queue or the operations of queue we have to create a queue. So let us look at how to create a queue. Here we are going to create a queue with a maximum size. What we are going to say is that an intqueue - intqueue here we are just indicating that the elements in the queue are all integers. You can have stringqueue, or you can have characterqueue. And the queue is equal to intqueue and with the maximum size and if queue is not equal to the null then you have to create a Head structure then you are creating the Head structure where you are going to put front is equal to -

1 and the queue rear is equal to -1. This depends and you can out it as 1 or 0 or -1 - it depends on what implantation you want to do. In this particular implementation we have chosen to put front and the rear equal to – 1 and we are putting the count of the queue equal to 0 and the maximum size has been set to max size. So this is how you create a queue so the creation of the queue is essentially what we do in this particular case - we say that the maximum size that we are allowing and we are saying that we are creating a node and we are creating two pointers front and rear then we are also counting the number of elements. It is not necessary always to count the number of elements but some times in the queue you don't have to maintain the count. In this particular example we are maintaining the count.

Enqueue operation: Now let go the next operation, the next operation is the enqueue operation. You remember the enqueue operation is an insertion operation we are going to insert the element as we have already discussed you always insert at the rear of the queue - there is no other choice you can only insert at the rear of the queue, because it follows the 'First In First Out' policy. So whoever comes first will be deleted first so whoever comes last will be inserted at the last. So first you'll check whether the count i.e. the number of elements in the queue is equal to the maximum size that is the maximum number of elements you are going to allow into the queue. This fixing up the maximum size is important in a queue because most applications of the queue as we have already seen require that there is a maximum limit for the elements in the queue. Even if it is a buffer there is a maximum limit to the size of the buffer - in a computer science application of the queue. So when the queue count is equal to the maximum value – the maximum size of the queue then you return it. The queue is full you cannot add any more elements or you cannot do an enqueue operation. Otherwise - Now you remember that the rear is pointing the last element in the queue, you increment the rear and then you put the queue rear – you check whether the queue rear is equal to the maximum size. If it is equal to the maximum size you are wrapping the queue that is now the queue can grow only from this side and if the queue rear is equal to 0 and then you are making the queue rear is equal to 0. Now you remember that this 0 is the starting point of the queue. If it is empty only you can add there - now we are assuming it to be 0 because the count is not equal to the maximum size. Then after you do this, these two are the conditions… here you are checking the maximum size, here you are checking whether you have reached the end of the queue, and if you have reached the end of the queue what you are doing is you are wrapping it round and then you come here - what you do is - in the array -- in that location of queue rear -you add whatever data you want to add- that is the data you want to input into the queue. If the queue count is equal to 0 then that means you are inserting into a null queue - means there is no element in the queue then you make the queue front equal to 1. You remember we have initialized to -1 so make it equal to 0 and make the queue count equal to 1 because you have one element in the queue. Otherwise if the count is not equal to 0 you just increment the count and return 1 – we return 1 just to say that the queue insertion have been successful. So please note in this particular implementation of enqueue what we are doing is we are fixing up the maximum size and we are checking here whether the count is equal to the maximum size. This means that the queue already contains as many elements as it can hold so you cannot insert. Then you are incrementing the rear pointer, then you are checking whether the rear is equal to the maximum size. Here is an important point to note – if the queue rear is equal to the queue maximum size it does not necessarily mean that the queue is full. Only if the count is also equal to the maximum size it is full. If this is the case it means that you have reached the maximum size. But then still there are some locations where you can insert that is why you are doing the wrap round okay. Then you are making the queue rear equal to 0 okay that means maximum size is been changed into 0.Then you are inserting the data element. Now here this part of the code is just to check whether this is a first element - whether you have inserted into an empty queue. Before this insertion happened - the count was 0 - now you are incrementing and making the count 1 - making the queue front to be equal to 0. Before it would have been -1 - now you are making it equal to 0. Otherwise, you are just incrementing the count and you are returning 1. Please note that the rear has already been incremented at this point and here you are incrementing the count. That is the normal queue operation. This is very important when we look at data structures... this looking at special conditions. This is a special type of queue where we are maintaining the count - you didn’t maintain the count you couldn’t do it this way okay. This is a special type of queue.

Linked list implementation: Now let us look at the linked list implementation. The previous one was enqueue implementation and in that enqueue implementation we were looking at the circular array which we have seen in the last lecture. Now here we are going to look at the linked list implementation. So if you look at the linked list implementation we have a node with front and you have a node with rear. Now in this case the front would be pointing to this and the rear would be pointing to this and you want to enqueue 8. So what you are doing is you are creating a node of the queue type and in that new node you are putting the value 8 and you are making the next equal to null and if front is equal to null - now if the queue was empty then you are making the front point to the node that you have just now created for 8. Otherwise you don’t change the front

- the rear. Next which would have been null is now pointed to CP and now the new rear is this CP. Now let me explain this again - now let this be equal CP. CP will have 8 and then it will have null and this null will now point to 8. Now this is how the enqueue operation is done. So look at this - 3, 7. This is the existing queue - now you want to insert 8 - so what you do is you create 8 - this is CP. The CP is having 8 and the pointer is made equal to null now this already existed and this you are creating now- this. Now this is equivalent to this step ok. Now this is your new queue and your front will point to this node and the rear is equal to CP that is the step. So this is how you do the enqueqe operation.
Now let us look at Dequeue. Again in the Dequeue operation - what we are doing is we are first looking at the Dequeue from the viewpoint of an array. So remember we have to delete an element from the queue - from where are you deleting- from the front of the queue. So initially you are checking whether the count exists - so if the count is equal to 0 then that means there is no element in the queue and the queue is empty. Otherwise what you do is - in this particular implementation of queue you are taking out the data whatever the element in the front you are taking it out. That is taken out and shown. The queue front is incremented to indicate that one element is deleted i.e. the front has been incremented and so this is the implementation. Now if queue front is equal to the queue max size - it means your front has reached the max size but remember your count will not be equal to max size so that means you have to again wrap round. So this is similar to what we did in enqueue and queue front is made equal to 0 and if the queue count is equal to 1 then it means - like in the previous enqueue that you are deleting the only element that is present in the queue that is you had a queue of one element- now you are deleting that element from the queue. So what you do is – that is this case - remember this was our creation condition where queue front and queue rear were made equal to -1 so that what you do and queue count is equal to -1 - this is just to say that the count is decreased by 1. So again this is similar to the enqueue operation that we did. First we are checking - dequeue operation in this case - first we are checking whether queue contains any element at all - if it contains - then you are outputting that element and you are incrementing the front. You are checking two things – you are just checking whether the front has reached the max size - if it has reached the max size then you have to wrap round – that is what you are doing here. The next condition you are checking is whether the queue is containing only one element which means after the dequeue operation the queue will now be empty. Now that is what you are checking here - now this is the dequeue operation.

Dequeue operation from the linked list viewpoint: Now let us look at the dequeue operation from the linked list viewpoint. Here again what you are doing is - if the front is null that is it doesn’t have any element -you cannot obviously do any deletion - so you are throwing an exception. Otherwise whatever is the result is that is whatever is in the front will be returned as a result. And your front will be pointing to the next node okay that is suppose you have three nodes – it will be pointing to this node - initially front is now pointing to this node now it has to point to this node. Now let see how this is done - now you have a queue with three values and this is what is there. Now you want to delete - you can only delete 8 so what you do is - the front and the rear are like this here - you can only delete 8, there is no other option. So after the deletion what happens - the front goes and this becomes the new front. So what we have done up to now - we have done is - we created a queue and we have enqueued using both arrays and linked list and we have used circular arrays with wrap round. We have done dequeue again with arrays and linked list and again in the array implementation of dequeue we have also done the wrap round. So this is the implementation of queue from the viewpoint of arrays and from the linked list view point.

Operations on Queues

Queue front operation: Now we are going to talk about some other important operations of the queue. So the first one we will talk about is the queue front. Look at the queue front operation. Please note that in the queue front operation, the status of the queue remains the same only the main thing is that we are accessing the front of the queue - so before we do that and we are outputting the value -even before we do that- we check whether there are any elements in the queue. So that is what we do here - so the count is equal to 0 - what we do is -we return 0. Else we just take out the data from the queue front position and return it. So this is a simple operation where we are just accessing the front element of the queue

Queue rear operation: So now let’s look at a similar operation that we do to access the rear of the queue. As you remember we can access the queue whether it is in the array implementation or a linked list implementation we can access two points in the queue - one is the front and the other is the rear. So what we do in this queue front and queue rear option is we are looking at accessing those two elements that are pointed out by front and rear.

Now here is a queue rear operation. Here we are checking whether the queue exists - if it does not exists we return 0 - otherwise you take out the element that is stored in the queue rear location and then you return that value and then return 1 so that is the simple operation as far as the queue is concerned. The other operations that you normally do on queue are the Boolean operations - where you check whether the queue is empty or check whether the queue is full so obviously we have done this checking already in the enqueue and in the dequeue operations. Queue empty we have checked in the dequeue operation so what we do is - we have checked whether the count is 0. Please note that this is the array implementations you are checking whether there are any more elements - if the count is 0 obviously the queue is empty. So if the count is equal to the maximum size - the queue is full. So these are Boolean operations - where we have returned the Boolean values.

Destroy queue operation: The next operation is the destroy queue operation. This is not to delete the elements one by one but to destroy the whole queue that is you don't want the queue to be there at all. If there exists a queue, you free the complete queue and if you cannot access the queue at all - the queue will not exist - so we return null. So just in a single step we are freeing the complete queue and returning the queue to the storage and we are returning null. Now this destroys the complete queue. This is not like the dequeue operation where we delete the elements one by one we just delete the whole queue. So these are the main operations that we have seen as far as queues are concerned - creating a queue, enqueueing that is inserting an element, dequeuing deleting from a queue, accessing the front and the rear elements of a queue and two Boolean operations -checking for queue full and queue empty condition and the next is we are destroying the whole queue.

Applications of Queues

Now let us look at some applications of queues. Now when we look at applications of queues we are going to look at the "Breadth First Search" first. This searching operation is very common in data structures and for many operations of computer science we need this searching operation. There are different methods of searching operation - one is depth first search and one is breadth first search. In this particular animation we are seeing here - we look at the breadth first search. What is important here is that we are using the queue as the data structure for implementing the breadth first search. So let us start - so first we have – we are starting with the root node and the children of the root node are inserted into the queue and then what happens because this is a queue operation this 2 - please note that the search value that we want to search for is 10 - that is the goal and the start is 1. So as soon as you see 1, the children of 1 are put here and then what happens is - you again when you search - the 2 is removed and 3 & 4 are incorporated then the children of 2 - 5 & 6 will be put here at the back of the queue. That is the operation of the First In First Out - but 3 was coming first - so next you traverse to 3 - that is from 1 you go 2 then to 3. Then what happens the children of 3 are removed and then the children of 3 are inputted but where will you put them - it will be put at the back of the queue. But according to the queue operation - next what you'll take out is 3. Now please note that we have travelled 1, 2, 3 and then we have travelled 4 because that is in the front of the queue so that is travelled and then the children of 4 – 4 doesn't have any children - so nothing is added next. You go to 5 - it is removed as it doesn't have children so nothing is removed then we move to 6 which is the next one. Now the children of 6 (9) will be added but it will be added at the back so that is what happens next. But what will you access next - that is 7 as it is in the front of the queue that is 7 comes next and then the children of 7 is put which is 10. But you still will not be able to access 10 because it is in the back of the queue, so next you go to 8 and then you go to 10. So what happens is that you have reached the front of the queue. So again just to understand 1, 2, 3, 4, 5, 6, 7, 8, 9 and then you will to the goal that is 10. The breadth first search - why we are talking about the breadth first search is because we are using queue as a data structure for implementing the breadth first search. It is a natural way of implementation because what you do is you start with a node and then it's children are put and you take the first node here and if the children comes then it comes at the back so when you want to finish of the nodes in one level you go to nodes in the next level and that is how the queue operation will be done - so this is the natural way of using the data structure for breadth first search.

Now let us go to the implementation of 'Breadth for Search': So now let us go to the implementation of 'Breadth first Search' - you have a location and you start - that location is first visited and you have a queue insert operation - if the queue is not empty you take out the front element and for the each unvisited neighbour N that is you have to put its children - enqueue its children into the queue and if the N reaches the goal then success is ensured otherwise what you do is you dequeue the element and then you go back. You repeat the while loop till the queue is not empty so what you are doing is as soon as you see a node you put it - take out the first element from the queue and then look at each unvisited node and see look whether it's children have been inserted into the queue and then keep removing nodes till there no more elements. This is the implementation of the BFS algorithm.

The priority queue: Now let us look at another topic that is very important in the queue that is the priority queue. Now in a queue it follows the normal 'First in First out Policy' however there are lot of applications of queues where you want to do some priority. You don't want always to insert only at the back but you want some elements to be inserted in the middle. The priority queue can work in two ways you do deletion as normally from the front of the queue but inserting don’t insert only at the rear of the queue but insert according to the priority. That is one way of implementing the priority queue. The other way is you always add at the back but when you are deleting don’t do it from the front but do it according to the priority. But the first method where you do insertion according to priority and deletion according - the normal queue that is how it is implemented. So what is a priority queue? It stores prioritized elements, there is no motion of storing at a particular position, and returns elements in priority order and the order is determined by the key in the element. So it is good that the order of return elements is not in the First in First out Policy or in the Last In First Out. You remember that First in First out Policy is followed for queue and Last In First Out policy is followed for stack and it is not random access also it is according to some - normally she should stand here but because of some priority she has been put here - so this is a typical example of a priority queue. There are lot of places where we use priority queue in computer science. We use priority queues to store values so suppose for example we have a priority queue of integers 3, 4, 7, and 10 - normally in a queue if you want to enqueue 8 it will come and sit here but in a priority queue let us assume that it is in sorted order 8 is inserted at this point and not at this point where it would have been done and then deletion you normally follow the normal queue delete - it is deleted from the front of the queue. So what is happening here is that the queue the elements - the priority here is that lowest element should be deleted first so that is the priority. So, it could also be the other way around – the highest valued elements could be deleted first then you can insert the element according to - here we are inserting the element in according to increasing order. Otherwise you can also insert in the decreasing order so it depends on what your application is. But here the important point to note is here when we are inserting into the queue or enqueueing. We are not enqueueing at the end of the queue or at the rear of the queue – we are inserting according to the key values - that is most crucial part. So the priority enqueue operation, inserts an element at an appropriate position of a priority queue according to the element’s priority. So this is what I have been explaining - so the priority dequeue operation deletes one element form the priority queue which is always the element with the smallest priority - this element with the smallest priority is returned by the priority dequeue. So examples are the plane landing managed by air traffic control, process schedule by CPU, college admission process for students. Now when you look at this plane landing managed by air traffic control - suppose you have some conditions - say depending on the fuel available in the plane, you want certain planes to land first - I mean earlier than the other planes. So don’t follow the first in first out you insert the elements - I mean the planes into the queue depending upon the amount of fuel left. This process scheduling for scheduling algorithm of the CPU one of the criteria they use is - if the job requires less time you give it more priority and then according to that you can queue it College admission is according to the marks that can be the criterion where according to the marks you choose the students so that is. So when you are standing in queue if a candidate comes whose rank is higher than yours he/she will placed in the front of the queue. So that is the typical example of priority queues.

Now let us look at some problems that we can do with queues. One important queue that we have not discussed but that is easily understandable is a Dequeue which is a double ended queue and this is a list from which the elements can be inserted or deleted at either ends. So this is called the dequeue but remember in a stack we can insert and delete an element only at one end, in a queue you can insert at one end and delete at another end you cannot do both at both ends. So Dequeue or double ended queue is similar to a queue but insertion and deletion can be done at both ends. So if you want to insert, you can insert from the front and if you want to delete you can delete from the front. Similarly if you want you can insert from the rear and delete from the rear so this is a double ended queue. So you can try to develop an array or a pointer implementation for a dequeue. It is quite a challenge – try to do that and you can also define a queue that enqueue , dequeue and onqueue this is slightly a different operation ...enqueue and dequeue already we saw. In the operation or the implementation that we saw for a circular array we can try simple array operation - without the count values and you can see how it works and there are different ways in which you can implement. But this onqueue operation is something different that we have not seen … this onqueue operation is a function which will – a Boolean function which just checks whether the element X is available in the queue which means it has to search the queue. This is a slightly different operation which we have not seen - so you try to implement that also. There are other implementations of the queue - another possible linked list implementation with no header cell okay? So you can out front = rear = nil so you can do it… and also you can look at the variant of the circular queue here - what we did was the circular queue was a wrap round - instead of that you can have a tag value and look at the implementation that way. There is a slight variation in the logic that you follow in this implementation. So this is how we have looked at queues. So basically in this lecture we have looked at the implementation of queues in general and we have looked at typical application of queue – the breadth first search where the queue seems to be a natural data structure for implementing the breadth first search and then we have looked at what is the priority queue. The priority queues sort of breaks the rules of queues but it is useful as it is not a random operation it gives some sort of structure and to do insertion or deletion from the queue according to some priority values.