The International Journal of Engineering and Science (IJES)
|| Volume || 6 || Issue || 7 || Pages || PP 48-51 || 2017 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

Queue with Improved Operation and Remove Constraint

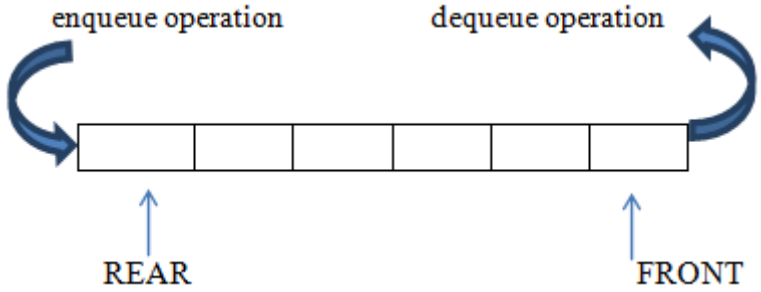
*Fakhruddin Amjherawala[1], Ummulbanin Amjherawala[2]
[1]*School of Computers, IPS Academy: Indore, Madhya Pradesh, India.*
[2]*School of Computers, IPS Academy: Indore, Madhya Pradesh, India.*
*Corresponding author: *FakhruddinAmjherawala*

----------**ABSTRACT**----------
*Queue is an abstract data structure is based on the rule of **FIFO.** Queue is used to insert and delete element in a proper sequence. In computer science it works as a CPU scheduling in an operating system as well as in real life on railway reservation counter and at a ticket counter of multiplex. Each of this show that element moved from their respective position while deletion occur, but the concept of queue shows that move front pointer to the position to delete an element. In this paper eliminate the operation of deleting an element at their respective position, explore the benefit to move the elements toward the starting positions of queue for further operation and also eliminate the constraint of overflow condition in the case when queue is full and remove first element and again try to insert next element into the queue.*
Keywords:*Dequeue, Enqueue,Front, Overflow, Rear,Underflow.*

Date of Submission: 13-07-2017 Date of Publication: 21-07-2017

I. INTRODUCTION

Queue [1] is an abstract data structure,somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e. the data item stored first will be accessed first.



enqueue() is the operation for adding an element into the Queue.
dequeue() is the operation for removing an element from Queue.

Figure: Queue Data Structure.

In this paper we show that removing element from front end but front never move to the position for remove an element.

II. PROBLEM

Queue is an abstract data structure is based on the rule of **FIFO.** While removing (dequeue) element from front end in the queue then move the front pointer to the respective position of an element. Due to this there is a conditional constraint occurs to insert an element at rear end and still there is a space to place next element after removing front element.
After deleting all the element Front and Rear pointer place at the end of queue then there is need to set them at the starting position for further operation.
In data structure removed above constraint by the implementation of circular queue, but in actual scenario queue never be align in circular fashion [2].

III. BACKGROUND KNOWLEDGE

Queue is a linear data structure which follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut(**FIFO**). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first. In a queue, we remove the item the least recently added.

Operations:

Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition. .
Front: Get the front item from queue.
Rear: Get the last item from queue.

```
Initially set Rear->-1 and Front->-1
int Dequeue(int queue[], Front){
int ele;
if(Front==-1){
      "Underflow occur"
Exit;}
   Front =Front+1;
ele = queue[Front];
return ele;}

Enqueue(int queue[], Rear){
if(Rear==Max-1){
      "Overflow occur"
Exit;}
Rear =Rear+1;
queue[Rear] = ele; }
```

IV. APPLICATION

Queue is used when things don't have to be processed immediately, but have to be processed in **F**irst **I**n**F**irst **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

a. When a resource is shared among multiple consumers. Examples include CPU scheduling Disk Scheduling.
b. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from front. If we simply increment front and rear indices, then there may be problems, front may reach end of the array. The solution to this problem is achieved by increase front and rear in circular manner in a circular Queue [3] as shown below.

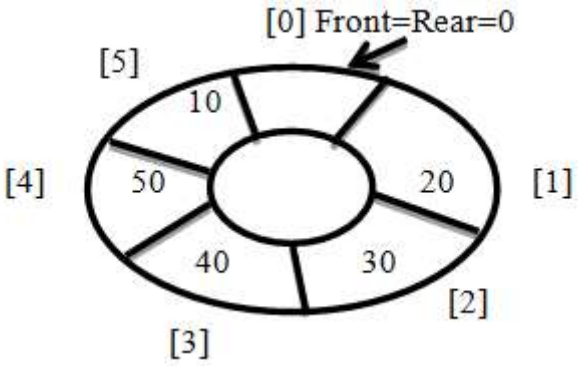


Figure: Circular Queue

```
Initially set Rear-> 0 and Front-> 0
Skip the first position for insertion
int Decqueue(int queue[], Front, Rear,Size)
{
int ele;
if(Front= =Rear){
        "Underflow occur"
          Exit;}
Front = (Front+1) % Size;
ele = queue[Front];
return ele;
}

Encqueue(int queue[], Rear, Size)
{
if(Rear==Max-1AND Front==0){
        "Overflow occur"
Exit;}
Rear = (Rear+1) % Size;
if (Rear=0) Rear=1;
queue[Rear] = ele;
}
```

V. RESEARCH OBJECTIVE AND APPROACH

To implement queue by eliminating the operation of deleting an element at their respective position and to move the elements toward the starting positions of queue for further operation and also eliminate the constraint of overflow condition in the case when queue is full by removing first element and again try to insert next element into the queue.

Front pointer will never move, it will always place at the starting position to delete an element. After deletion, the remaining elements shift from its respective position toward the front position of queue.

In this case deleting any single element from the queue, it will never meet overflow condition as in the case of simple queue.

In this paper we will implement how Rear move ahead for inserting an element (Enqueue) as well as move back one position after deleting front position element, but Front fixed at the starting position for performing deletion (dequeue).

While implementing this way queue will never be implementing via circular manner.

VI. DESIGN OF THE ALGORITHM AND RESULT

In this approach follow the concept of queue in FIFO manner but in a different manner which are as follows:

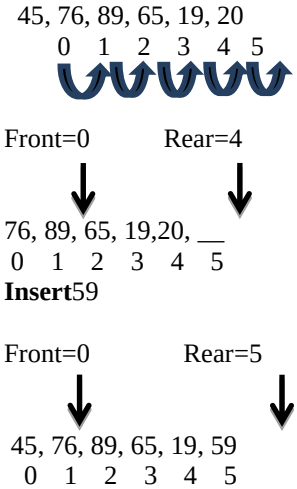
```
Initially set Rear->-1 and Front->0
int Dequeue(int queue[], Rear, Front)
{
int ele;
if(Rear<Front){
        "Underflow occur"
Exit;}
ele = queue[Front];
//shift queue element from right to left
for(int i=Front; i < Rear; i++)
queue[i]=queue[i+1];
// make rear position available for insertion
Rear = Rear-1;
return ele;
}
```

```
Enqueue(int queue[], Rear, Front)
{
if(Rear==Max-1){
       "Overflow occur"
Exit;}
    Rear = Rear+1;
queue[Rear] = ele;
}
```

Example:
Queue has 6 element spaces.
After insertion following are the element of queue.

Deletion of first element then insert element in the existing queue not possible it shows "overflow condition", but in our approach we can insert next element into the queue.

Case:
Delete front element 45 then shifting occur of each element one by one towards front position.



follow the above procedure and achieve the result.
This paper eliminate the existing queue dequeue() problem which makes toswitch towards circular queue.

VII. CONCLUSION

Time complexity of all operations of existing queue like enqueue(), dequeue() is O(1). There is no loop in any of the operations, but it stops the operation of insertion after deleting a single element. In our case there is a loop in dequeue() and its complexity become O(n) but it eliminate the constraint "overflow " and insertion occur without switching toward circular queue.

REFERENCES

[1]. A. M. Tenenbaum, Y. Langsam and M. J.Augenstein, Data Structure Using C (India,Pearson Education, 1990).
[2]. J. P. Tremblay, P. G. Sorenson,An Introduction to Data Structures with Applications (New York,Tata McGraw-Hill,1976).
[3]. S. Chattopadhyay, D. G. Dastidar, M.Chattopadhyay, Data Structure Through C Language (India,BPB,2001).

Fakhruddin Amjherawala. "Queue with Improved Operation and Remove Constraint." The International Journal of Engineering and Science (IJES) 6.7 (2017): 48-51.