Tutorial Link https://codequotient.com/tutorials/Queue:Introduction/5a563aa5c83e417a5c00dbf3

TUTORIAL

Queue: Introduction

Chapter

1. Queue: Introduction

Topics

Queues are a very useful data structure in Computer Science. Queue is the most common data structure which allows elements to be inserted at one end called Rear and deleted at another end called Front. Queue is also known as FIFO data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed.

A queue is a linear structure in which element may be inserted at one end called the rear, and the deleted at the other end called the front. Figure below pictures a queue of people waiting for their turn. Queues are also called First-In First-Out (FIFO) lists. An important example of a queue in computer science occurs in a timesharing system, in which programs with the same priority form a queue while waiting to be executed.

Figure: People waiting for their turn

Computer science also has common examples of queues. Our computer laboratory has 30 computers networked with a single printer. When students want to print, their print tasks “get in line” with all the other printing tasks that are waiting. The first task in is the next to be completed. If you are last in line, you must wait for all the other tasks to print ahead of you.

In addition to printing queues, operating systems use a number of different queues to control processes within a computer. The scheduling of what gets done next is typically based on a queuing algorithm that tries to execute programs as quickly as possible and serve as many users as it can. Also, as we type, sometimes keystrokes get ahead of the characters that appear on the screen. This is due to the computer doing other work at that moment. The keystrokes are being placed in a queue-like buffer so that they can eventually be displayed on the screen in the proper order.

Operations on Queue

Insertion

Addition of new element in queue is known as insertion or EnQueue operation on Queue. Whenever an element is added to the queue, the value of REAR is increased by 1; this can be implemented by the assignment

```
REAR = REAR + 1
```

For example, following is adding some elements in the queue.

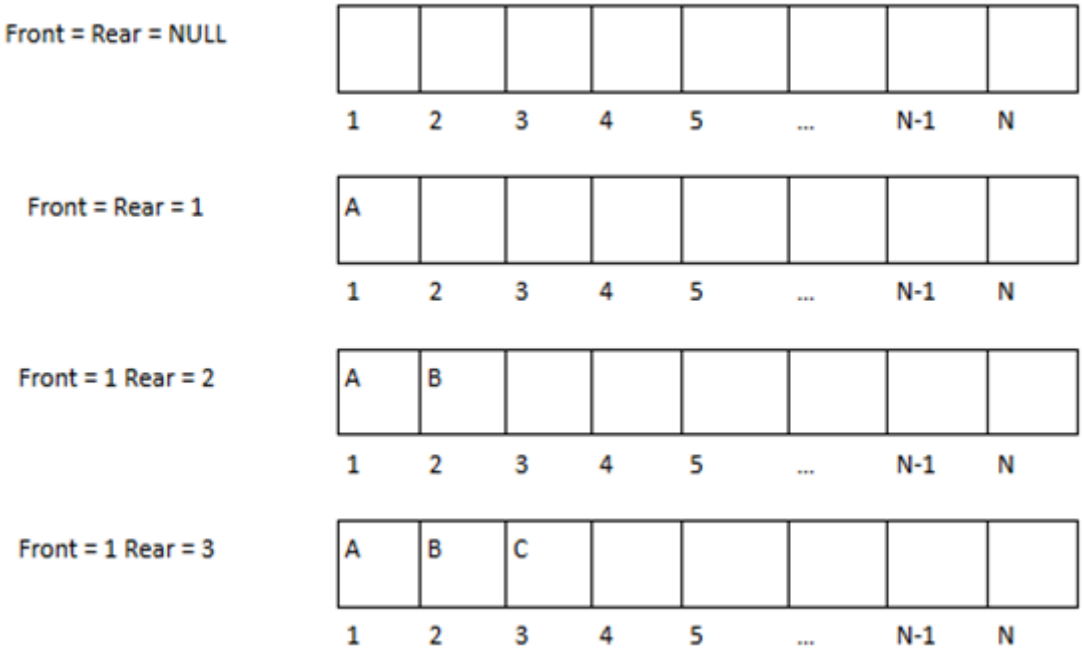

Figure: Insertion in a Queue

As indicated in the figure above, when there is no element, then front and rear both are equal to NULL. If an element is inserted, then front and rear both becomes 1. If another item is inserted, the rear changes to 2, and after another insertion rear becomes 3.

The operation of adding an item into a queue is implemented by the following algorithm

```
Algorithm : INSERT (QUEUE, N, FRONT, REAR, ITEM)
1. If REAR == N then
        Write OVERFLOW and Exit.
2. If FRONT == NULL, then
        FRONT = 1 and REAR = 1.
     Else
        REAR = REAR + 1.
3. QUEUE [REAR] = ITEM.
4. Exit.
```

In this algorithm the first step checks for the available space for inserting a new element. If no space is available, then the overflow condition occurs. Otherwise control flows to the second step, in which location where new element is to be inserted is found. In the third step, actual insertion is made.

Deletion

Whenever an element is deleted from the queue, the value of FRONT is increased by 1; this can be implemented by the assignment

```
FRONT = FRONT + 1
```

Suppose that our queue contains only one element, i.e., suppose that

```
FRONT = REAR = 1
```

and suppose that the element is deleted. Then we assign

```
FRONT = NULL and REAR = NULL
```

to indicate that the queue is empty and this operation can be depicted by figures below.

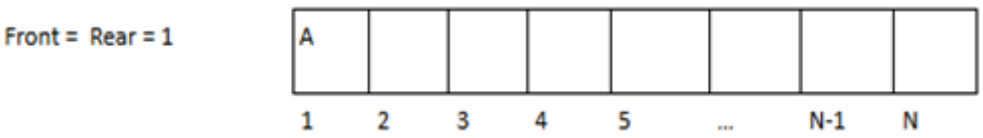

Figure: A queue in which FRONT = REAR = N

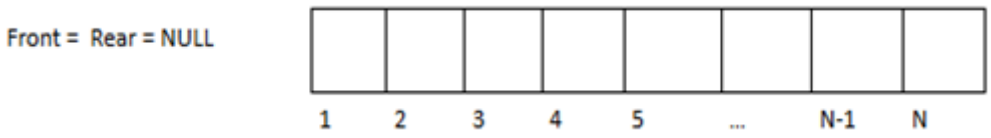

Figure: After deleting an element in the queue

The operation of removing an item from a queue is implemented by the following algorithm

```
Algorithm : DELETE (QUEUE, N, FRONT, REAR, ITEM)
1. If FRONT = NULL then
         Write UNDERFLOW and Exit.
2.     ITEM = QUEUE[FRONT].
3.    If FRONT = REAR then
         FRONT = NULL and REAR = NULL.
Else
```

```
    FRONT = FRONT+1.
Return ITEM
Exit
```

In the first step, it is checked whether any element is existing in the list or not. If no element is available in the queue, then underflow condition occurs. In the second step the element at the FRONT is saved in a variable. In the next step the element is actually deleted.

Computer Representation of Queue

Queues may be represented in the computer in two ways i.e. by means of linear arrays and one-way linked lists. First we will be considering representation of queues with the help of arrays.

Queue using Array:

Queues will be maintained by a linear array QUEUE and two pointer variables: FRONT, containing the location of the front or first element of the queue; and REAR, containing the location of the rear or last element of the queue. The condition FRONT = REAR = 0 will indicate that the queue is empty. Following is the implementation of Queue in a program: -

Javascript

```
class CQQueue{
    constructor(){
        this.SIZE = 100;
        this.array = new Array(this.SIZE).fill(0);
        this.front = -1
        this.rear = -1
    }

    enQueue(data){
        if (this.rear == this.SIZE)   // Queue is full
            return;
        if(this.front == -1 && this.rear == -1){
            this.front++;
            this.rear++;
```

```
        }else
            this.rear++;
        this.array[this.rear] = data;
        console.log(`${data} enqueued to queue`);
    }

    deQueue(){
        if (this.front > this.rear){
            console.log("Queue is Empty.");
            return -1
        }
        let item = this.array[this.front];
        this.front++;
        console.log(`${item} dequeued from queue.`);
        return item;
    }
}


function main(){
    let ob1 = new CQQueue();
    ob1.enQueue(10);
    ob1.enQueue(20);
    ob1.enQueue(30);
    ob1.enQueue(40);

    ob1.deQueue();
    ob1.deQueue();
    ob1.deQueue();
    ob1.deQueue();
    ob1.deQueue();

    ob1.enQueue(50);
   return 0;
}

main()
```

C

```
#include <stdio.h>
#define SIZE 10
```

```
int front=-1, rear=-1;
int array[SIZE];

// Method to add an item to the queue.
void enqueue(int item)
{
  if (rear == SIZE)   // Queue is full
    return;
  if(front == -1 && rear == -1){
    front++;
    rear++;
  }
  else
    rear++;
  array[rear] = item;
  printf("%d enqueued to queue.\n",item);
}

// Method to remove an item from queue.
int dequeue()
{
  if (front > rear)
  {
    printf("Queue is Empty.\n");
    return -1;
  }
  int item = array[front];
  front++;
  printf("%d dequeued from queue.\n",item);
  return item;
}

int main(){
  enqueue(10);
  enqueue(20);
  enqueue(30);
  enqueue(40);
  dequeue();
  dequeue();
  dequeue();
```

```
  dequeue();
  dequeue();
  enqueue(50);
}

```

Java

```
class QueueArray
{
  static int SIZE=10;
  static int front=-1;
  static int rear=-1;
  static int array[]=new int[SIZE];

  // Method to add an item to the queue.
  void enqueue(int item)
  {
    if (rear == SIZE)   // Queue is full
      return;
    if(front == -1 && rear == -1){
      front++;
      rear++;
    }
    else
      rear++;
    array[rear] = item;
    System.out.println(item+" enqueued to queue.");
  }

  // Method to remove an item from queue.
  int dequeue()
  {
    if (front > rear)
    {
      System.out.println("Queue is Empty.");
      return -1;
    }
    int item = array[front];
    front++;
    System.out.println(item+" dequeued from queue.");
    return item;
```

```
  }
}

class Main{
  public static void main(String args[]){
    QueueArray ob1=new QueueArray();
    ob1.enqueue(10);
    ob1.enqueue(20);
    ob1.enqueue(30);
    ob1.enqueue(40);

    ob1.dequeue();
    ob1.dequeue();
    ob1.dequeue();
    ob1.dequeue();
    ob1.dequeue();

    ob1.enqueue(50);
  }
}

```

Python 3

```

class CQQueue:
    def __init__(self):
        self.SIZE = 100
        self.array = [0]*(self.SIZE)
        self.front = -1
        self.rear = -1

    def enQueue(self,data):
        if(self.rear == self.SIZE):
            print('Queue Full')
            return
        else:
            if(self.front == -1 and self.rear == -1):
                self.front+=1
                self.rear+=1
            else:
                self.rear+=1
```

```
            self.array[self.rear]=data
            print(data,'enqueued to queue')

    def deQueue(self):
        if(self.front > self.rear):
            print('Queue is Empty')
            return -1
        else:
            item = self.array[self.front]
            self.front+=1
            print(item,'dequeued from queue.')
            return item

if __name__ == '__main__':
    queue = CQQueue()
    queue.enQueue(10)
    queue.enQueue(20)
    queue.enQueue(30)
    queue.enQueue(40)

    queue.deQueue()
    queue.deQueue()
    queue.deQueue()
    queue.deQueue()
    queue.deQueue()

    queue.enQueue(50)
```

C++

```
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;

class CQQueue{
    int SIZE = 100;
    int front = -1;
    int rear = -1;
    int *array;
    public:
        CQQueue(){
```

```
            array = new int[SIZE];
        }

        void enQueue(int data){
            if (rear == SIZE)   // Queue is full
                return;
            if(front == -1 && rear == -1){
                front++;
                rear++;
            }else
                rear++;
            array[rear] = data;
            cout<<data<<" enqueued to queue.\n";
        }

        int deQueue(){
            if (front > rear){
                cout<<"Queue is Empty."<<endl;
                return -1;
            }
            int item = array[front];
            front++;
            cout<<item<<" dequeued from queue.\n";
            return item;
        }
};


int main(){
    CQQueue ob1;
    ob1.enQueue(10);
    ob1.enQueue(20);
    ob1.enQueue(30);
    ob1.enQueue(40);

    ob1.deQueue();
    ob1.deQueue();
    ob1.deQueue();
    ob1.deQueue();
    ob1.deQueue();
```

```
53
54      ob1.enQueue(50);
55     return 0;
56  }
```

Video Solution

<iframe width="560" height="315" src="https://www.youtube.com/embed/Q0N1ICIni90" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>