**TECHNO INTERNATIONAL NEW TOWN**



**( (Formerly known as Techno India College of Technology)**

**Block – DG,Action Area 1,New Town,Kolkata -700156,West Bengal,India**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**TECHNICAL REPORT WRITING FOR CA-2 EXAMINATION**

**TOPIC: - How to efficiently implement k Queues in a single array?**

**Data Structure and Algorithm PCC-CS301**

**(Paper Name) (Paper Code)**

**Name: Sandip Babu**

**Section: A**

**Roll No: L-01**

**Registration No:**

**Semester: 3rd**

**Year: 2022 – 2023**

**Date of Submission: 02.09.2022**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_INTRODUCTION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**The Queue data structure is an ordered collection of items wherein the addition of new items happens at one end called the rear, and the deletion of new items happens at the other end called the front.**This ordering principle is called First-In-First-Out or FIFO.

This report will discuss one such important question: Implementation of K queues in a single array. It is crucial to understand that if there are K queues in a single array, then each queue should satisfy the following conditions:

Insertion of new elements will always happen from the rear end of the queue.

Deletion of an element will always happen from the front of the queue.

While inserting and deleting elements in a queue, a continuous check for overflow and underflow needs to be done.

Another important point to keep in mind while designing a data structure for implementing K queues in a single array is that the queue is open at both ends. One end of the queue is open for insertion, and the other end of the queue is open for deletion operation.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_METHODLOGY\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

In this report, we have explained how to implement K queues in a single array. The space of the array should be utilized optimally by the K queues.

As we could see in this report implementing 2 queues in an array required implementing 2 push and pop methods for each queue in the array. A logical approach to solve this is to use another parameter k representing the queue that needs to be modified and sending it to the push and pop methods. But that will not be sufficient since we do not know exactly the heads of the k queues. So, we must calculate them. Since we cannot have a hard coded variable, the only way is to use an array that will store the indexes for the queues.

The simple way to do that is to see if there are remained elements that don't belong to any queue, and that because an array might have even or odd numbers of elements. The number of them can be simple to find by using the mod function and then increase the index of the first rest number of queues by 1.

Size queue is = 5/3=1=>each queue will have 1 element, SizeRest = 5 mod 3=2=>unallocated elements will be allocated for first 2 queue by increasing them with 1 element

For example:

having an array of 6 elements and 3 queues will mean that each queue will have 2 elements with the head indexes for the queues [2,4,6]

having an array of 5 elements and 3 queues will mean that not all the queues will be equals. We will have in this case 2 elements unallocated, so the last queue will have with 1 element less than the first ones having the head indexes for the queues [2,4,5]

The downsize of this approach is that it is space inefficient, meaning the size of the queues are fixed and if we want to insert one element in a full queue we cannot do that even though there are free spaces in other queues.

**Algorithm**

The algorithm for push and pop methods will be the same as in the mentioned article the only difference consisted in calculating the indexes for each queue.  
For that we are going to use 2 arrays:

one for storing the heads of each queue

one for storing the movement of elements inside each queue

**Pseudo code for calculating the indexes of the k queues**

if k > 0 and sizeArray >= k

sizeQueue = sizeArray / k;

sizeRest = sizeArray mod k;

array idxQk[k] that will store the moved index of the k queue

array idxQh[k] that will store the head index of the k queue

i = 0;

loop

if i < sizeRest

idxQk[i] <- sizeQueue +1;

idxQh[i] <- idxQk[i];

i <- i+1

end loop

i = sizeRest;

loop

if i < k

idxQk[i] <- sizeQueue ;

idxQh[i] <- idxQk[i];

i <- i+1;

end loop

**Time & Space complexity**

For calculating indexes for k queues the time and space will be O(n).  
For push method the time will be O(1) since we are doing one operation.  
For pop method the time will be O(n) since we are moving elements.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CONCLUSION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

The best part of the above implementation is, if there is a slot available in the queue, then an item can be enqueued in any of the queues, i.e., no wastage of space. This method requires some extra space. Space may not be an issue because queue items are typically large, for example, queues of employees, students, etc where every item is of hundreds of bytes. For such large queues, the extra space used is comparatively very less as we use three integer arrays as extra space.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_REFERENCES\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**https://tutorialspoint.dev/data-structure/queue-data-structure/efficiently-implement-k-queues-single-array**](https://tutorialspoint.dev/data-structure/queue-data-structure/efficiently-implement-k-queues-single-array)

[**https://www.geeksforgeeks.org/efficiently-implement-k-queues-single-array/#:~:text=A%20simple%20way%20to%20implement,size%20of%20array%20be%20n**](https://www.geeksforgeeks.org/efficiently-implement-k-queues-single-array/#:~:text=A%20simple%20way%20to%20implement,size%20of%20array%20be%20n)**.**

**https://www.codingninjas.com/codestudio/library/how-to-efficiently-implement-k-queues-in-a-single-array**