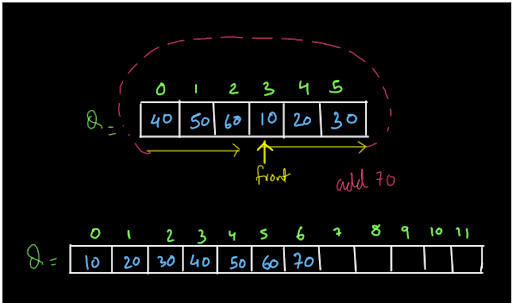
**1. Problem Discussion:**

****

The queue is starting from index 3 and ending at index 2 and is completely filled. Now, if you remember from the normal queue, if we try to add 70 to this queue it will print "queue overflow". But, we have to change this. We have to add such functionality to the queue such that whenever it gets filled and we want to add a new element, it doubles its total capacity and the queue also starts from index 0 and we get a lot of space free to insert new elements.

We have to build a queue of ours with the same features and implementations of the queue data structure that we have studied in the INTRODUCTION TO QUEUES video. We recommend you watch the question video to understand the question completely and get a better idea of what exactly we need to do.

**2. Approach :**

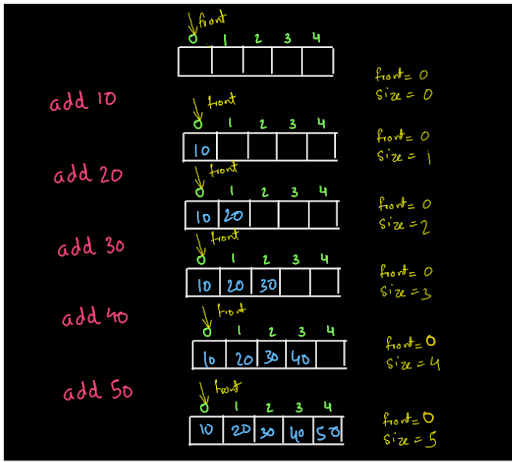
Dear reader, we recommend you solve the NORMAL QUEUE problem first before moving on with this problem as we are just going to add a small functionality in our already written function. You may refer to the NORMAL QUEUE SOLUTION VIDEO if you have any doubts regarding it.

Building a Normal Queue:

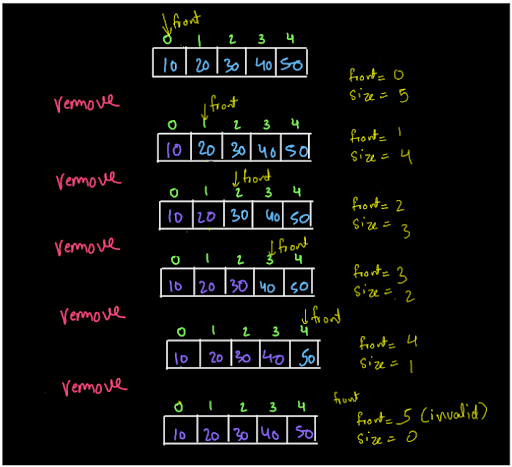
You may skip this section of the article if you already know how to build a normal queue. If you want to revise it or even study it, you are most welcome. Let us start with a very basic idea of building a linear queue.

Building a Linear Queue:

As we know the queue data structure works in a FIFO (first-in-first-out) fashion. So, we add the data at the end of the queue every time and we remove the data from its front. Have a look at the figure shown below:

****

We keep on adding the data in the queue and data gets added at the last of the queue. Now, if we try to add any more elements to the queue we will not be able to add those elements as the queue is full. The condition for the queue to be full is when this array "data" becomes full. So, the condition is size=data.length. So, we are adding the values into the queue always at the rear end. Let us call the index at which value is inserted as rear. So, rear=front +size. For instance, initially front is at 0 and the size is also 0. So, rear=front+size=0+0=0, and we add the value 10 at data[rear]. Then after adding a value, the size becomes 1. So, the next value is added at rear=front+size=0+1=1. So, data[rear]=data[1]=20. We can continue inserting like this. Similarly, the removal of elements from the queue is shown below:

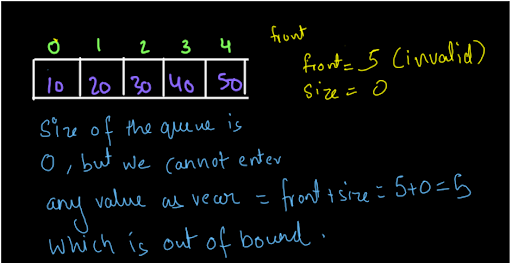
****

We always remove the first element from the queue. The elements shown in purple indicate that they are not a part of the queue anymore. For instance, we remove 10 from the queue, not by shifting the elements, just by moving front to the next index (and decrementing the size value by 1) indicating that the queue now starts from index 1 and any element before index=1is not a part of the queue. When size=0, there is no element left in the queue and we cannot dequeue (remove) any element from the queue. So, if we try to remove any element after this, we will print "queue underflow".

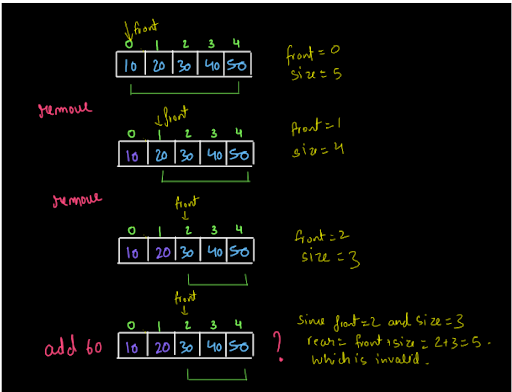
Did you notice anything strange in this queue? Hint: look at (fig-3) the last array, when the queue is empty.

Need for a Circular Queue:

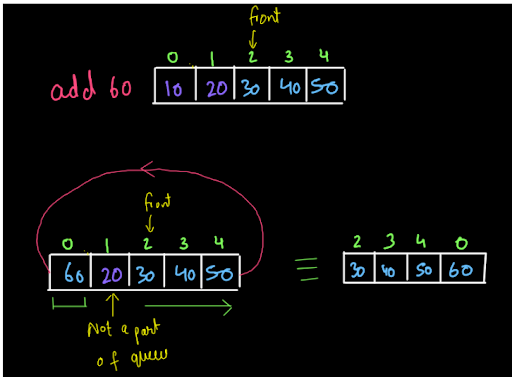
Let us take the last stage of the queue as shown in fig-3.

****

The size of the queue is 0 as we have removed all the elements from the queue. Still, we can not enter any value into the queue as the front became 5 when we removed the last value and also rear=front+size=5+0=5, which is also an invalid position to enter a value into the queue. Similarly, look at the diagram shown below:

****

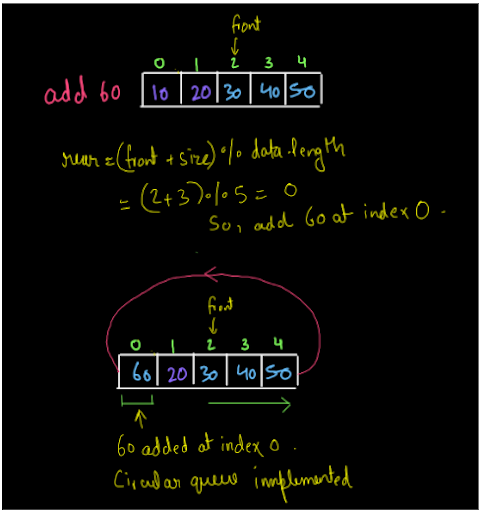
We removed the first two elements from the queue. So, only three elements inside the array are now a part of the queue. Now, if we want to insert an element into the queue, it can be inserted at the rear position only. Now, rear=front+size=2+3=5, which is out of the bound for the array. So, even if 2 spaces were vacant in that array i.e. 2 spaces were available which were not a part of the queue, still, we could not insert our elements there. So, you saw, in the first case, when we had the entire array vacant then also, we were not able to enter any values into it and we were not able to enter the values into the queue even if certain places (not the entire array) were vacant. So, what is the solution to this? Have a look at the diagram given below:

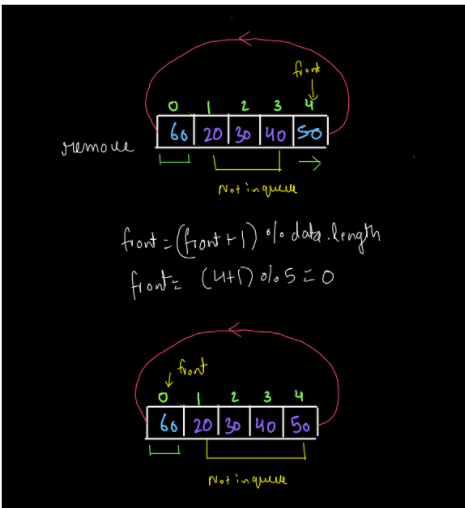
****

We want our queue to show a circular behavior like this to utilize the space available. So, this can be implemented easily, using a modulus operator.

Building Circular Queue:

Enqueue (Add): While adding the values into the queue, we were calculating the rear as rear=front+size. Now, we will calculate rear as rear=(front+size)%data.length. This will help us achieve a circular implementation of the queue. (See fig-7) Dequeue (Remove): While removing the values, we will not just move front forward by doing front++. Rather, we will do: front=(front+1)%data.length. This will help achieve the circular implementation (See fig-8). Display: While displaying the queue also, we will display it from the front. So, we will again increment the front as shown above i.e. front=(front+1)%data. length

****

****

**3. Code Discussion:**

1• We will modify the add(value) function that we have created in our previous problem. 2• Whenever we get our queue full i.e. when size=data.length, we will create another array that will be double the size of the queue. We will copy our elements from the queue to the array in such a way that the elements of the queue are placed linearly starting from index 0 in the array as shown in fig-1. For this we will do arr[i]=data[(front+i)%data.length] for n iterations, where n is the current size of the queue. 3• After this, we will copy the array back to the original queue and we will add the element at the end as we used to do for a normal queue.

**4. Code For Normal Queue:**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class CustomQueue {

int[] data;

int front;

int size;

public CustomQueue(int cap) {

data = new int[cap];

front = 0;

size = 0;

}

int size() {

return size;

}

void display() {

for (int i = 0; i < size; i++) {

int idx = (front + i) % data.length;

System.out.print(data[idx] + " ");

}

System.out.println();

}

void add(int val) {

if (size == data.length) {

System.out.println("Queue overflow");

} else {

int idx = (front + size) % data.length;

data[idx] = val;

size++;

}

}

int remove() {

if (size == 0) {

System.out.println("Queue underflow");

return -1;

} else {

int val = data[front];

front = (front + 1) % data.length;

size--;

return val;

}

}

int peek() {

if (size == 0) {

System.out.println("Queue underflow");

return -1;

} else {

int val = data[front];

return val;

}

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

CustomQueue qu = new CustomQueue(n);

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("add")) {

int val = Integer.parseInt(str.split(" ")[1]);

qu.add(val);

} else if (str.startsWith("remove")) {

int val = qu.remove();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("peek")) {

int val = qu.peek();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size")) {

System.out.println(qu.size());

} else if (str.startsWith("display")) {

qu.display();

}

str = br.readLine();

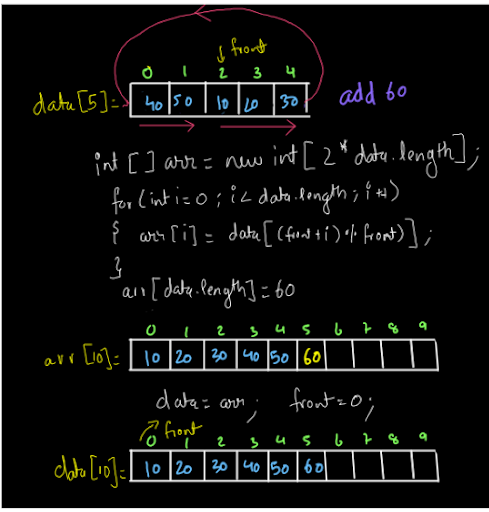
}

}

}

**5. Building Dynamic Queue:**

So, as discussed in the question, we now need to add the feature of dynamic size in the queue. For that, we will modify the add(value) function of the queue written above. We will take an array whose size will be double the size of the current array. Then, we need to fill in the values from the queue to the array. For that, we will use the same logic as used in the display function. We will iterate from i=0 to i=n-1, where n is the size of the queue. We will fill in the values from the queue to the array by applying the following logic: arr[i]=data[(front+i)%data.length]

****

The queue is such that the front is at index=2 and the rear is at index 1. So, it is in a circular fashion. Now, we want to add 60 to this queue. But there is no space in the queue. So, we make an array "arr" of size twice the current size of the queue. So, it now has enough space to accommodate n elements more, where n is the current size of the queue. We copy the elements from the queue to the array in such a way that the ordering of the queue is maintained. After this, we add an element at index=n i.e. arr[n]=60, and then we do data=arr so that the changes are being made to the original queue. Also, do not forget to change the front of the queue to zero as the new front =0. So, this is what we are going to do. We hope you got this, It is a very simple procedure. So, let's make the required changes to our add(value) function.

**6. Code:**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class CustomQueue {

int[] data;

int front;

int size;

public CustomQueue(int cap) {

data = new int[cap];

front = 0;

size = 0;

}

int size() {

return size;

}

void display() {

for (int i = 0; i < size; i++) {

int idx = (front + i) % data.length;

System.out.print(data[idx] + " ");

}

System.out.println();

}

void add(int val) {

if (size == data.length) {

int[] arr = new int[2 \* data.length];

for (int i = 0; i < size; i++) {

int idx = (front + i) % data.length;

arr[i] = data[idx];

}

data = arr;

front = 0;

}

int idx = (front + size) % data.length;

data[idx] = val;

size++;

}

int remove() {

if (size == 0) {

System.out.println("Queue underflow");

return -1;

} else {

int val = data[front];

front = (front + 1) % data.length;

size--;

return val;

}

}

int peek() {

if (size == 0) {

System.out.println("Queue underflow");

return -1;

} else {

int val = data[front];

return val;

}

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

CustomQueue qu = new CustomQueue(n);

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("add")) {

int val = Integer.parseInt(str.split(" ")[1]);

qu.add(val);

} else if (str.startsWith("remove")) {

int val = qu.remove();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("peek")) {

int val = qu.peek();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size")) {

System.out.println(qu.size());

} else if (str.startsWith("display")) {

qu.display();

}

str = br.readLine();

}

}

}

So, dear reader, if you have any doubts regarding the above procedure or the code you may refer to the complete solution video to clear your doubts if any.

**7. Analysis**

Time Complexity :

The time complexity of the above procedure will be O(1) if we have to insert an element just at the end without increasing the size and O(n) if we have to increase the size as we will have to copy all the values of the queue to the array. Since the question is about creating a dynamic queue, the time complexity for creating a dynamic queue is O(n) .

SPACE COMPLEXITY :

The space complexity is also O(n) as we make an array of size=2n where n is the current size of the queue. Dear reader, we hope that you got the above time and space complexity analysis also. With this, we have completed this problem.

Suggestions:

Here are some suggestions from our side that you do not want to miss:

1• This was a very easy problem if you had solved the previous question. So, it is very important to maintain the order in which the questions are given so that you do not face any difficulties in the upcoming questions and build concepts in perfect order. 2• Yes, we have a queue available in Java directly and we can use it to solve our problems but it is very important to understand the working and physical structure of any data structure so that we can implement it easily later on. So, understand this concept wisely and revise it once too. Until then, Happy Coding!!!!