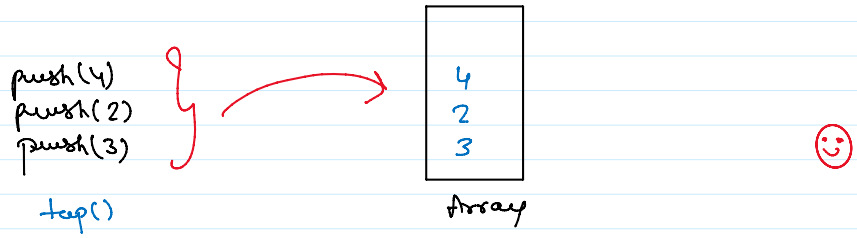
SQP2 Queue Implementation Using Array

Saturday, November 12, 2022 4:28 PM

In this problem we implement the queue all operation using Array data structure

Queue

→ Now we know that the Queue is the data structure that follows the (FIFO) Concept.



Important Point

Now Critical we know that the top element is (4) but this is the Queue Implementation and we know that its work on (FIFO) Concept

→ So according to the (FIFO) my top element is (3)

pop()

→ Delete the first element from the array that we insert into the array last time just opposite to the Stack

→ In Java Implementation we required same more Variable

~~#front~~

that pointing to the front element into the array

Rear

that pointing the last element / and own count of Index

and push the element on

(rear%n)

→ that is the own implementation index where we insert the own element

Time Complexity = O(1)

→ because we used only a single operation at a one time for a single digit (constant)

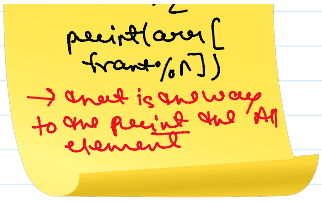
Space Complexity = O(N)

→ used for storing the own Queue element into the array

Now How to print the hole queue element

```
for(i=front;
  rear-1){
   print(arr[
    front%n])
```

→ that is the array



Implementation Python

```
# gfg: https://practice.geeksforgeeks.org/problems/implement-queue-using-array/1
class Queue:

    def __init__(self):
        self.queueSize = 10000
        self.front = 0
        self.queue = []


    def push(self,x):
        if len(self.queue) < self.queueSize:
            self.queue.append(x)

        return -1

    def pop(self):
        if len(self.queue) > 0:
            x = self.queue[self.front]
            self.queue.remove(self.queue[self.front])
            return x

        return -1

    def top(self):

        if len(self.queue) > 0:
            return self.queue[self.front]

        return -1

    def size(self):
        return len(self.queue)


    def isEmpty(self):
        return len(self.queue) == 0


ans = Queue()
ans.push(4)
ans.push(14)
ans.push(24)
ans.push(34)
print(ans.top())
print(ans.size())
print(ans.pop())

print(ans.size())
print(ans.top())
```

→ This is the way to implement queue using the Array data structure

Implementation Java

```
import java.util.ArrayList;
import java.util.*;
import java.util.Stack;

//gfg: https://practice.geeksforgeeks.org/problems/implement-queue-using-array/1
public class L2_Implement_Queue_Using_Array {
    public static void main(String[] args) {
        System.out.println("L2_Implement_Queue_Using_Array");



        Queue q = new Queue();
        q.push(4);
        q.push(14);
        q.push(24);
        q.push(34);

        System.out.println("The peek of the queue before deleting any element " + q.top());
        System.out.println("The size of the queue before deletion " + q.size());
        System.out.println("The first element to be deleted " + q.pop());
        System.out.println("The peek of the queue after deleting an element " + q.top());
        System.out.println("The size of the queue after deleting an element " + q.size());
        System.out.println("The isEmpty of Qeueu " + q.isEmpty());
        System.out.println("Hole ans " + q.getans());


    }
}

class Queue{

    int queueSize = 10000;
    int[] queue = new int[queueSize];
    int front = 0;
    int rear = 0;
    int count = 0;

    void push(int x){
        if (count < queueSize) {
            queue[rear%queueSize] = x;
            rear+=1;
            count+=1;
        }
        else{
            System.out.println("Queue if Full");
        }
    }

    int pop(){
        if(front == rear || count == 0){
            return -1;
        }
        int x = queue[front % queueSize];
        queue[front % queueSize] = -1;
        front+=1;
        count-=1;
        return x;
```

```
        front+=1;
        count-=1;
        return x;

    }

    int top(){
        if(front == rear || count == 0){
            return -1;
        }
        return queue[front];
    }

    int size(){
        return count;
    }

    boolean isEmpty(){
        return front == rear;
    }

     List<Integer> getans() {
        List<Integer> ans = new ArrayList<>();

        for(int i = front;i < rear;i++){
            ans.add(queue[i%queueSize]);
        }

        return ans;
    }
}
```