Ticket system documentation.

Overview

Create a program that can be used for a ticket system. We are implementing the problem statement using a queue method, where elements can be inserted at any time but only one element that has been in the queue the longest can be removed. We will be using the above-mentioned principle to ensure the person who entered the window first gets the ticket first.

Algorithm

1. **Define call and Initialization**

    1.initialise the w windows with n size

    i. For i in range(w)
    ii. For j in range (n)
    iii. A[i][j] <-- None

    2.Initialise status of window counter

    iv. For i in range(w)
    v. A[i] <-- False
    vi. A[i] <-- True

2. **Define Add Person Function**

    1.find the available window

    i. Call enqueue method
    ii. Update status

3. **Define Give Ticket**

    1.find the windows with persons

    1.call dequeue

    2.update Status

4.Define update status

i. For i in range(w)
ii. Check whether queue is empty or not

If empty

A[i] = False

Else

A[i] = True

4. **Define isopen**
    i. Return open status of particular window
5. **Define Getwindow**
    i. Return all elements of particular window
6. **Define main**
    1.define object
    2.call add person
    3.call give ticket

Time complexity

Function Name	Worst case complexity
init	O(n*w)
isOpen	O(1)
getWindow	O(n)
addPerson	O(w)
giveTicket	O(w)
updateOpenStatus	O(w)

Alternate solution

We can use a linked list instead of a queue using array implementation. In linked list implementation we will be using two pointers, front, and rear. The front points to the first item of the queue and rear points to the last item.

- enQueue() - This operation adds a new node after rear and moves rear to the next node.
- deQueue() - This operation removes the front node and moves front to the next node.

Time complexity will be less cause both enqueue and dequeue will have a time complexity of O(1) as it only changes a few pointers in both operations.