**Design PS16Q1**

# Defined a BoxOffice class as below –

class BoxOffice:

def \_\_init\_\_(self, w, n):

# queue size

self.n = n

# number of windows

self.w = w

# Initialize as many queues as the number of windows

self.queues = [ Queue(j, n) for j in range(w) ]

# Initialise all windows as closed except 1st window

self.windows = [ False for i in range(w) ]

self.windows[0] = True

The problem is solved using a Queue ADT. The BoxOffice class has an array of Queue object and a 1:1 matching windows list associated with each Queue object. Class Queue is defined as

class Queue:

def \_\_init\_\_(self, win, n):

self.size = n

self.item = [ None for i in range(n) ]

self.cnt = 0

self.front = 0

self.rear = 0

self.win = win

Design makes used of a special ‘cnt’ variable as well to follow the queue size. This helps us detect when the queue is full. If we check ‘front’ == ‘rear’ this can indicate full empty or completely full conditions. So either we keep one index empty in the queue item list or we chose to instead use a ‘cnt’ variable to distinguish the conditions. If ‘cnt’ == ‘n’ we know the queue is full.

Functions implemented in Queue ADT –

1. def isEmpty(self)
2. def getSize(self)
3. def enqueue(self, pid)
4. def dequeue(self)
5. def getFront(self)
6. def getQueueElems(self)

All the above functions for the queue ADT above takes O(1) time except **getQueueElems.**

**getQueueElems**

* loops through the complete queue from front to rear
* add each person id found in order to a python list and
* returns the same

In worst case scenario, the queue might be full with ‘n’ people already waiting, so it will have to do complete one loop from front to rear which would be position 1..n. Hence the time complexity for this function is O(n).

BoxOffixe object implements the following functions:

1. def isOpen(self, win)
2. def getWindow(self, win)
3. def addPerson(self, pid)
4. def giveTicket(self)

isOpen - simply indexes into the winow id and checks if it is True for open or False for closed. So this runs in O(1) time

getWindow - directly indexes into the queue for the specified window and call queue[index].getQueueElems(). This functions as discussed above in Queue ADT takes O(n) time in worst case. Hence the complexity of getWindow() is O(n).

addPerson -

* loop through all open windows and choose the one with smallest queue size
* Also makes sure that if two windows are having same size, the one with smallest index (window ID) will be used
* worst case this will loop through all windows and find them all open and full.
* In the same single pass, also store the first encountered closed window index so that we can directly use that in case all open windows are full
* Enqueue the person to either the opened window with shortest queue size or open a new window and enqueue to that window queue. Either way this is O(1) as discussed earlier

Since this at most has to check all window queue sizes, one loop from 1..w is needed and hence the time complexity for the function is O(w)

# Validation

A detailed input validation routine is written to validate the input file format and all parameters are correct. Any error is captured in the prompts **promptsPS16Q1.txt**

Also, all debug output if any is captured in the promptsPS16Q1.txt file as well. All stdout prints are disabled.

Prompts file was not needed for this question as per the professor Sheetal. It is added only for stdout error prints while debugging and can be ignored. There is no stdout prints when program is executed and output file is generated with final output.

# Output

Output is captured in **outputPS16Q1.txt** file. Sample input file is **inputPS16Q1.txt**

# Logging

Added a logger implementation which logs into the prompts and output file. It has capability to simultaneously print in console or in the specified files. By default console logging is commented.