Data Structures and Algorithms Design

Academic Year 2021-2022

Design Document - Assignment 1 – PS06 - [Admission Ticket]

**Authors**

Vinaya K

Muthuraja M

R Balaji

**Contribution Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Name (as appears in Canvas)** | **ID NO** | **Contribution** |
|  | Vinaya K | 2021C104200 | 100% |
|  | R Balaji | 2021C104109 | 100% |
|  | Muthuraja M | 2021C104112 | 100% |

**Problem Statement:**

The head of office administration in a college needs your help to implement an admission ticket distribution system from the office for the upcoming university exam. He has a list of specific tasks in how the distribution system should work. Below are the tasks

* There is only ‘c’ number of office counters in the admin section. Each counter can have at max ‘n’ number of students waiting in line.
* To start with, only one counter is opened. If the number of students waiting in line in that counter exceeds n, then the next counter is opened, and students can join the line in that counter. Likewise, if both the first and second counters have n number of students waiting in each queue, then a third counter is opened. This can go on until the maximum number of counters c is reached. Let us assume that once a counter is opened it never closes. A new counter is only opened if all open counters are full. When an admission ticket is issued to the student, the count of the number of students in each open queue is reduced by 1.
* When a new student has to join the queue, the system has to prompt him/her to join a queue such that they are issued admission ticket as fast as possible. The system prompts the student based on these factors
  + First it looks for an open counter with the least number of students and prompts that counter number. If more than one counter has the least number of students, then the system can prompt the student to join the first counter (smaller counter Id) it encounters with the least number of students.
  + If the queues of all open counters are full and a new counter can be opened, then the new student is prompted to join the new queue for the new counter.
  + If all queues for all counters are full, a corresponding message is displayed. That student need not be considered in the next iteration.

● After a queue is prompted to a student, the student or system cannot change the queue

**Overview of the program:**

The system/application is designed using Python 3.7 and using queue data structures.

The number of counters that can be opened and the maximum size of queue in each counter is captured from the input file “InputPS06.txt” with the key word “AdmissionticketSystem”

Ex: AdmissionticketSystem:5:8 implies that a maximum of 5 counters can be opened and each counter can have a maximum of 8 students in the queue.

The program is written to handle the following scenarios:

1. Adding a student to the queue
2. Issue an Admission Ticket in each of the open counters
3. To list the counter contents
4. To check whether the counter is open/close

**Scope:**

In-scope:

Issuing an admission ticket to the students as assigned to the counter as per the system design.

Out of Scope:

Removing a student from the queue once assigned to a counter. Once a student is assigned to a counter the only functionality available is to issue an admission ticket.

**Assumptions:**

1. The input file and output file will be located in the default python folder
2. If the number of students to be added to the queue and the counter exceeds the possible sum of all counters the students are not assigned to any queue.

Ex: We have 3 counters and max 5 students to be added in each queue and the input file contains instructions to add student #16 and above will be ignored by the system.

**Flow Chart:**

Start

**No**

**Yes**

**Yes**

End

Is end of file

Print Output File

**No**

End

**AddStudent**

**GiveAdmissionTicket**

**IsOpen**

**GetCounter**

Assign a counter to the student

Return the queue contents of the counter

Issue an admission ticket to the student

Return the counter status

Process keyword from the input file

Read the next line

Initialize the counters and arrays

Is “AdmissionTicketSystem”

Process keyword from the input file

Read Input file

**Time Complexity Analysis:**

**Initialize:**

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

self.c = c

self.n = n

self.queues = [[None for i in range(n)] for j in range(c)]

self.starts = [0 for i in range(c)]

self.ends = [0 for i in range(c)]

self.sizes = [0 for i in range(c)]

self.open = [False for i in range(c)]

self.open[0] = True

This function is iterated through nested for and the time complexity would O(n\*m), if instead of m, we had to iterate on n again, then it would be O(n^2)

**Time Complexity is O(n^2)**

**IsOpen:**

def isOpen (self, counterId):

return self.open[counterId-1]

This function is has only one statement and as a constant number binds the entire function the time complexity is represented as O(1),

**Time Complexity is O(1)**

**GetCounter:**

def getCounter (self, counterId):

return self.queues[counterId-1]

This function is has only one statement and as a constant number binds the entire function the time complexity is represented as O(1),

**Time Complexity is O(1)**

**AddStudent:**

def addStudent (self, studentID):

fullCounter = True

min = 0

for i in range(self.c):

if(self.sizes[i] != self.n):

fullCounter = False

break

if fullCounter:

return -1

for i in range(self.c):

if(self.open[i] and self.sizes[i] < self.sizes[min]):

min = i

if(self.sizes[min] == self.n):

min = self.openNextCounter()

if(min != -1):

self.queueEnqueue(min,studentID)

return min;

The IF block has a time complexity of O(n log n) (that’s common runtime for efficient sorting algorithms). The ELSE block has a runtime of O(1).

**Time Complexity is O(n)**

**OpenNextCounter:**

def openNextCounter(self):

fullCounter = True

for i in range(self.c):

if not self.open[i]:

fullCounter = False

return i;

if fullCounter:

return -1

This function is iterated “c” times based on the inputs passed. Since c is an integer the time complexity is represented as O(n)

**Time Complexity is O(n)**

**GiveAdmissionTicket:**

def giveAdmissionTicket (self):

numberOfTicketsIssued = 0;

for i in range(self.c):

if self.open[i]:

numberOfTicketsIssued += self.queueDequeue(i)

return numberOfTicketsIssued

Time Complexity: This function is iterated “c” times based on the inputs passed. Since c is an integer the time complexity is represented as O(n)

**Time Complexity is O(n)**

**Enqueue:**

def queueEnqueue(self, counterId, studentID):

# Check queue is full or not

if(self.sizes[counterId] == self.n):

# Insert element at the rear

else:

self.open[counterId] = True

self.queues[counterId][self.ends[counterId]] = studentID

self.ends[counterId] = (self.ends[counterId] + 1) % self.n

self.sizes[counterId] += 1

This function is has only if and assignment statements without any loopings and runtime can be represented as a constant number, the time complexity is represented as O(1),

**Time Complexity is O(1)**

**DeQueue:**

def queueDequeue(self, counterId):

isDequeued = 0

# If queue is empty

if(self.sizes[counterId] == 0):

isDequeued = 0

# Pop the front element from list

else:

self.queues[counterId][self.starts[counterId]] = None

self.starts[counterId] = (self.starts[counterId] + 1) % self.n

self.sizes[counterId] -= 1

isDequeued = 1

return isDequeued

This function is has only if and assignment statements without any loopings and runtime can be represented as a constant number, the time complexity is represented as O(1),

**Time Complexity is O(1)**

**Acceptance Criteria:**

The following test scenarios are verified

|  |  |  |
| --- | --- | --- |
| **Test Scenario** | **Description** | **Sample Output (desired)** |
| Adding a Student | Process the input file and assign a student to the open counter | addStudent:1 >> c1  addStudent:2 >> c1 |
| Counter Status | Check whether the counter is Open and return the status as “True” if the counter is open else as “False” | isOpen:1 >> True  isOpen:1 >> False |
| Give Admission Ticket | Issue an admission ticket to the student and remove the student from the counter queue | giveAdmissionTicket: >> 2 giveAdmissionTicket: >> 1 |
| Get Counter | List all the elements in the counter queue for all the open counters.  Return empty if the counter is closed. | getCounter:1 >> [1, 2, 3, 4, 5] getCounter:1 >> [None] |