COMP47500 – Advanced Data Structures in Java

Assignment No: 1
Date: 24th February, 2023

Title: A Queuing Model of the Airport Departure Process

Assignment Type of Submission:			
Group	**Yes/No**	**List all group members' details:**	**% Contribution Assignment Workload**
	Yes	Student Name: Vivek Murarka Student ID: 22200673	20
		Student Name: Nikhitha Grace Josh Student ID: 22200726	20
		Student Name: Purvish Shah Student ID: 22200112	20
		Student Name: Ravi Raj Pedada Student ID: 22200547	20
		Student Name: Meghana Kamsetty Ravikumar Student ID: 22200568	20

1. Problem Domain Description

The situation of airplanes having to wait in the runway queue for takeoff and landing has become a significant problem due to the required separation time between them. This waiting time leads to increased fuel consumption. To address this issue, an approach known as arrival/departure manager (AMAN/DMAN) has been suggested and put into operation at some of the world's major airports. However, the effectiveness of AMAN/DMAN is heavily dependent on the specific actions and algorithms it employs. Therefore, ongoing efforts to update and improve the algorithm are necessary to enhance the overall system's performance.

A queuing model for the airport departure process was suggested by Ioannis Simaiakis and Hamsa Balakrishnan at MIT. In this paper, they presented a queue-based approach through which we can observe the congestion at airports or predict taxi-out time for individual flights.

The main focus of this paper is to anticipate airport performance on the assumption that the pushback schedules are already known. However, the paper does not explore how the uncertainty in the pushback schedules could affect the airport's performance.

The Model:

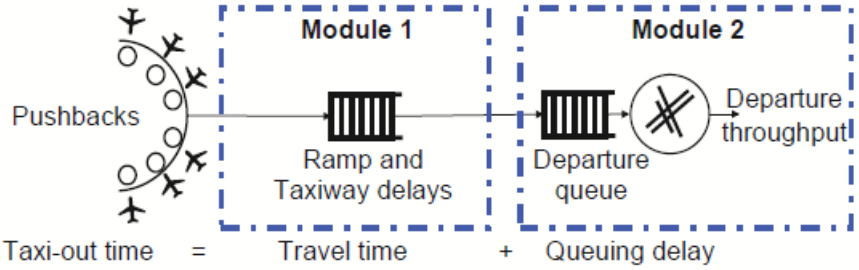


Figure 1: Departure Process Model

Figure 1 illustrates the proposed model, which comprises two separate modules. The first module covers the process that occurs between the time when aircraft are pushed back from their gates and travel to the runway. The second module focuses on the queuing process that occurs at the departure runway.

By modeling the departure process in this manner, the taxi-out time T_l of each departing aircraft l can be expressed as

$$T_l = T_l^{\text{travel}} + D_l \qquad (1)$$

where T_l^{travel} is the travel time of each departing aircraft l from its gate to the departure runway, and D_l is the queuing delay that aircraft l experiences.

Within the framework of this model, a pushback refers to the specific moment when an aircraft is physically separated from its assigned gate. Additionally, the taxi-out time is defined as the duration between the pushback and the actual takeoff of the aircraft.

Aircrafts leave their gates as per the pushback schedule and move onto the ramp and taxiway system, and taxi to the departure queue which is formed at the threshold of the departure runway. While travelling, the aircrafts interact with one another.

T_l^{travel} travel time of aircraft from their gates to their departure runway, is given by

$$T_l^{\text{travel}} = T_{unimpeded} + T_{taxiway} \qquad (2)$$

Here,
$T_{unimpeded,}$ the unimpeded taxi-out time refers to the expected amount of time it would take for an aircraft to taxi out without any obstacles or delays. The Federal Aviation Administration (FAA) has defined this as the optimal taxi-out time that would occur when there is no congestion, inclement weather, or other factors that might cause the aircraft to be delayed during its journey from the gate to takeoff, as stated in the FAA's 2002 guidelines.

The Office of Aviation Policy and Plans of the Federal Aviation Administration (FAA) in 2002 employed a linear regression analysis to establish the unimpeded taxi-out time by examining the recorded taxi-out times along with the departure and arrival queues.

This average unimpeded time for an airline at a given airport for a particular period can be downloaded from FAA Operations & Performance Data.

The second term, $T_{taxiway}$, of equation(2), represents the delay that results from interactions with other aircraft that are travelling on the ramp and taxiway en route to the departure queue.

The paper suggests a linear equation to evaluate $T_{taxiway,}$ given by,

$$T_{taxiway=} \alpha R(t) \qquad (3)$$

The parameter α is determined by the airport and the layout of the runway, and it signifies the estimated frequency of times that an aircraft will come to a halt due to congestion, multiplied by the average duration of each stop. The likelihood of an aircraft stopping rises in proportion to the number of other aircraft present on the ramp and taxiway system en route to the departure queue at the time when the aircraft starts its pushback.

And $R(t)$, is given by the quantity of aircraft that are departing and travelling on the ramps and taxiways towards the departure queue at the start of period t.

Combining equation(2) and (3) with (1), we get,

$$T_l^{\text{travel}} = T_{unimpeded} + T_{taxiway}$$

$$T_l^{\text{travel}} = T_{unimpeded} + \alpha R(t)$$

$$T_l = T_l^{\text{travel}} + D_l$$

$$T_l = T_{unimpeded} + \alpha R(t) + D_l$$

Another factor that we must consider is the time taken by the aircraft to travel from the departure gates to the runway queue. Further the gate from the runway more time it will take to travel, hence our final equation to generate the expected wheel off time is given by

$$T_l = T_{unimpeded} + \alpha R(t) + D_l + T_{\text{gate}}$$

Queueing Model:

For the aircraft to depart from the airport, it needs to travel through two queues.

First is the taxiway queue. Once an aircraft is pushed back from the gates, it enters the ramp and taxiway queue. This is a dynamic queue and doesn't have a fixed length. As more aircrafts enter this queue the size of the queue increases. There could be more than one plane which can be pushed back at the same time. This creates a confusion as to which aircraft enter this queue first and how long it needs to stay in this queue before it enters the second queue.

Once the aircraft enters the taxiway queue it goes through several stages of test which make it flight ready and then only starts heading to the departure queue. Some aircraft might take less time than others. There are other factors also which affect the time taken by the different airline carriers. Hence, some airlines might take less time than others. The mean time taken by different airlines at a given airport in different seasons is recorded by the FAA and data is publicly available.

We consider this time along with the average time taken by aircraft to cover from its gate to queue, to prioritize which aircraft enters the queue first. There could be some aircrafts already in this first queue which might have more expected wheel-off time than the incoming aircraft queue. Hence, we need to prioritize this new incoming aircraft in the queue depending on its expected wheel-off time. Post this, push all the existing aircraft in the queue which have more expected wheel-off time and then recalculate their new expected wheel-off time, depending on the size of the queue.

The second queue is the departure queue, which is of finite length. If there is space available in the departure queue, then the aircraft with the highest priority in the taxiway queue leaves the taxiway queue and enters the departure queue. Aircraft in the departure queue are served on a First-Come-First-Served (FCFS) basis. If any aircraft ahead in this queue could not depart from the runway at its expected time, it will delay the aircraft behind this. This is considered a queueing delay.

We record this queuing delay of each aircraft by subtracting the actual wheel-off time from the initially expected wheel-off time. The mean queueing delay gives the efficiency of the airport in handling the departure traffic. And this observation can be used to further reduce the congestion at the airport.

2. Theoretical Foundations of the Data Structure(s) utilised

To process the queueing model discussed earlier, we need two types of Queue data structures, first is the priority queue for taxiway queue and then a fixed size queue for the departure queue.

1. Queue

A queue is a linear data structure in which elements are inserted at one end (rear) and removed from the other end (front). It follows the FIFO (First In First Out) principle, which means that the first element inserted in the queue is the first one to be removed.

We use this queue to represent and process the departure queue in our queuing model. This queue is of finite length, as there is a limited waiting area for an aircraft at the runway queue. The aircraft which entered first in this queue departs first, thus adhering to the FIFO principle.

A linked list can be used to implement a queue data structure, which is called a linked list queue or simply a queue using linked list. In a linked list queue, each element of the queue is represented as a node in the linked list.

The front of the queue is represented by the head of the linked list, and the rear of the queue is represented by the tail of the linked list. When an element is enqueued, a new node is added at the tail of the linked list, and when an element is dequeued, the node at the head of the linked list is removed.

The advantage of using a linked list as a queue is that it allows dynamic resizing of the queue as elements are added or removed. Linked list queue operations such as enqueue and dequeue have a time complexity of O(1) only when the head and tail pointers are properly maintained. Otherwise, they can take O(n) time in the worst case.

2. Priority Queue:

A priority queue is a way to organize data so that the item with the highest priority can be accessed quickly and efficiently. It works similarly to a regular queue, but with the added feature of each element having a priority value assigned to it. In this type of queue, elements are arranged in order based on their priority value, with the highest priority element always placed at the front of the queue. As elements are removed from the queue, the next highest-priority element moves to the front.

Different types of data structures can be used to implement priority queues, depending on the specific needs of the application, including arrays, linked lists, binary heaps, and balanced binary search trees. Priority queues are especially useful in situations where items need to be processed in order of priority, such as in job scheduling, network routing, and simulation systems.

To create a priority queue for storing data of aircraft currently in a taxiway queue, we opted to use a double linked list. This is because the length of the queue may vary and we require the ability to frequently prioritize and reorder the data as new aircrafts are added.

During aircraft insertion, we prioritize them based on their wheelOffTime and update the queue accordingly. This means that every time an aircraft is inserted, all other aircraft behind it must be reevaluated and their wheelOffTime recalculated. This ensures that the aircraft queue remains properly sorted according to priority, with the aircraft with the earliest wheelOffTime being positioned at the front of the queue.

We initially designed our algorithm to traverse the linked list from the head in order to find the correct position for the new aircraft. While doing so, we would recalculate the wheelOffTime for the rest of the aircrafts in the queue up until the tail node. This approach had a time complexity of O(n) in both the best and worst cases.

However, in order to optimize this algorithm, we modified it to traverse the linked list from the tail node instead. While traversing, we would simultaneously recalculate the wheelOffTime for the aircraft in the queue until the appropriate position was found for the new aircraft to be inserted. This updated approach resulted in a significant improvement in the best case, with a time complexity of O(1), while maintaining a worst case time complexity of O(n).

Input:

Because of the nature of the data sensitivity, the flight data is not publicly available and can be requested from the FAA using appropriate credentials. Hence, we have data with 200,000 records generated randomly for the following fields.

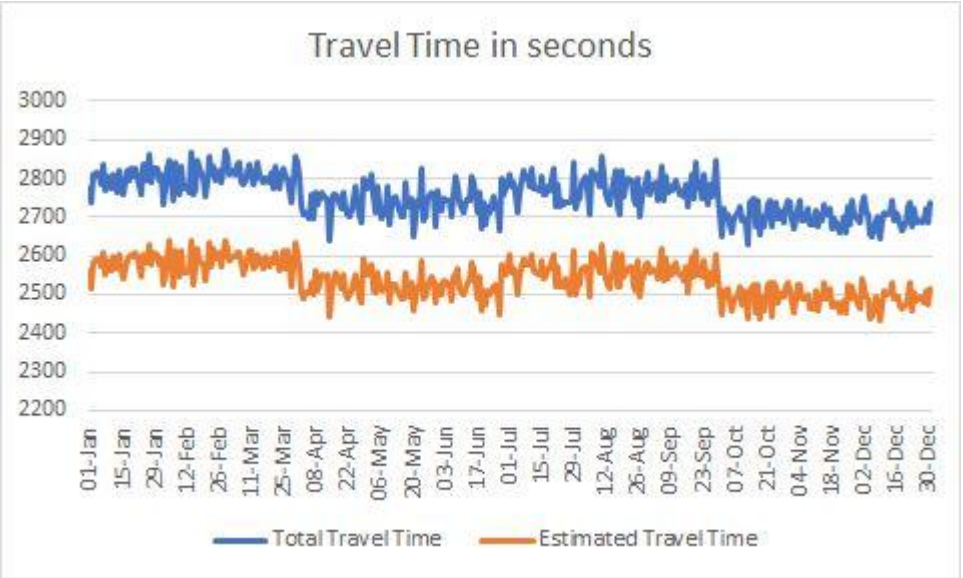
1. Unimpeded Data: This data is downloaded from the FAA website. We have analyzed data for the year 2012 of the JFK airport.
2. Departure Date: For each date we are generating random entry for the airline which operates in JFK airport.
3. Airline code: Airline that operated in 2012 at JFK airport.
4. Season: Unimpeded time for each airline is estimated by the FAA based on 4 seasons.
5. Gate no: Gate from which aircraft is pushed back. For generalization we have randomly assigned gates to each aircraft from 1 to 20.
6. Taxi-out time: The duration an aircraft takes ideally to travel from gate to runway queue.
7. Push back time: The time when the aircraft was pushed back from the gate.
8. Delay: The delay which each aircraft experienced at the runway just before its departure , due to various factors, which gives the actual wheel-off time.

Output:

The purpose of the queuing model is to study the congestion at the Airport. The congestion for each day can be observed by calculating the average difference between the actual wheel off time and expected wheel off time.

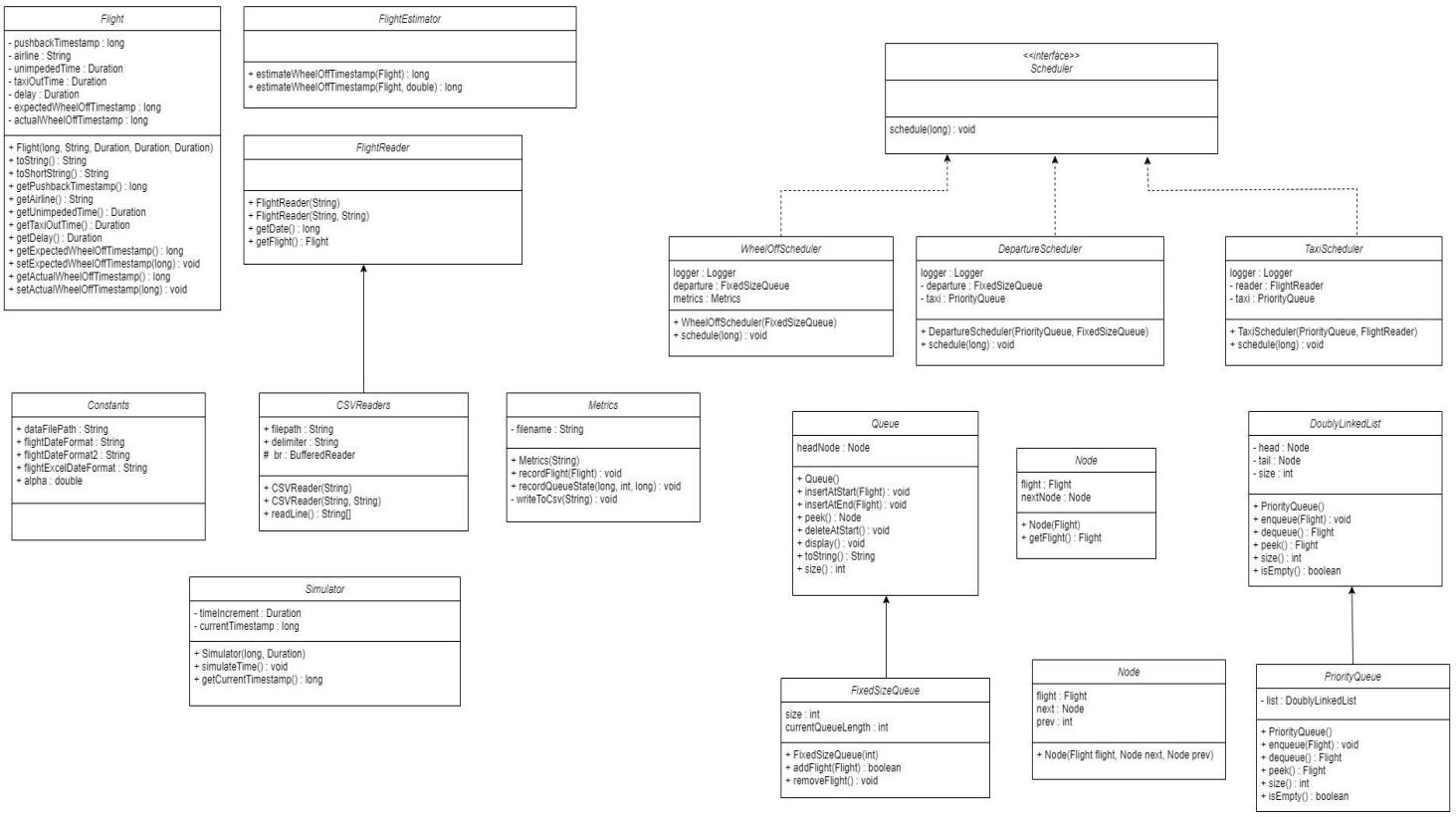
We calculated this difference for over 200,000 entries. The time taken to simulate the entire model was 45 seconds.

The difference between wheeloff time and actual wheel off time can be studied from this graph.



It can be observed that the average delay that each aircraft experienced was in between 250~300 seconds. This delay can be further reduced so that airport efficiency can be increased.

3. Analysis/Design (UML Diagram(s))



4. Code Implementation

GitHub (link): https://github.com/black-hawks/airport_departure_queuing

5. Set of Experiments run and result

1. Analysis of doubly linked list

   When implementing a simple queue for queuing aircraft in a taxiway, it is important to consider the potential downside. While a simple queue is efficient for handling first-come-first-serve scenarios, it may not be the optimal approach for situations where aircrafts have varying travel times to the runway.

   If we simply enqueue aircraft as they arrive without considering their travel time, it is possible that an aircraft with a shorter travel time may end up being delayed due to congestion caused by aircraft with longer travel time ahead of it in the queue. This could lead to inefficient use of resources and potentially even missed flight schedules.

In order to address the issue of varying travel times and potential congestion in a simple queue, we implemented a priority queue that prioritizes aircraft based on their wheelOffTime. This allows us to enqueue aircraft in a way that ensures the ones with earlier wheelOffTimes are given priority and can reach the runway more quickly.

To insert a new aircraft into the priority queue, we traverse the queue until we find the appropriate position for the new aircraft to be inserted. Once we have found the position, we calculate the new wheelOffTime for all the aircrafts that come after the position where the new aircraft was inserted. This ensures that the priority queue remains properly sorted and the order of the aircraft is optimized for minimizing delays and congestion.

While this approach provides a more efficient queuing system than a simple queue, it does come at the cost of increased time complexity. The time complexity for inserting a new aircraft into the priority queue is O(n), which means that the time required to enqueue aircraft may increase as the size of the queue grows.

To further optimize the priority queue, we implemented it using a double linked list data structure. This allows for efficient insertion and removal of aircraft while maintaining the priority order based on the wheelOffTime.

In addition to using a double linked list, we updated the enqueue algorithm to traverse the linked list from the tail rather than the head. This approach allows us to efficiently find the appropriate position for a new aircraft while simultaneously recalculating the wheelOffTime for the other aircraft in the queue.

By traversing the linked list from the tail, we are able to minimize the number of aircraft for which we need to recalculate the wheelOffTime, resulting in a best-case time complexity of O(1). However, in the worst-case scenario where the new aircraft needs to be inserted at the head of the queue, we still need to traverse the entire linked list, resulting in a time complexity of O(n).

Overall, the use of a double linked list and the updated enqueue algorithm allows us to efficiently maintain a priority queue for aircraft in a taxiway, taking into account their varying travel times and minimizing delays due to congestion. By carefully considering the trade-offs between time complexity and performance, we are able to design a queuing system that meets the needs of the airport or airline operation while ensuring efficient use of resources.

2. Improvement in the equation proposed by the paper for calculating expected wheeloff time.

   As discussed earlier to calculate expected wheel off time we use following equation:

   $$T_l^{\text{travel}} = T_{unimpeded} + \alpha R(t)$$

   Where $R(t)$ is the number of aircrafts ahead of it.
   To calculate $R(t)$ we need to calculate the queue length, and if the initial expected time is less than the expected time of any aircraft already in the taxiway queue, we need to shift that aircraft further in the queue.

   To avoid overhead of calculating queue length we enhanced this equation by using following analogy

   Travel time for first aircraft, $T_1^{\text{travel}} = T_{unimpeded} + \alpha R(t)$, *where* $R(t)=1$
   Travel time for Second aircraft, $T_2^{\text{travel}} = T_{unimpeded} + \alpha R(t)$, *where* $R(t)=2$
   Travel time for Third aircraft, $T_3^{\text{travel}} = T_{unimpeded} + \alpha R(t)$, *where* $R(t)=3$
   Travel time for Third aircraft, $T_4^{\text{travel}} = T_{unimpeded} + \alpha R(t)$, *where* $R(t)=4$

   Now, if fourth aircraft has less initial expected wheel off time than 2nd aircraft, such that,

   $$T1 < T4 < T2 < T3$$

   Then, we need to recalculate new T_3^{travel} and T_4^{travel}, so the new T_3^{travel} and T_4^{travel} can be given by:

   $$T_3^{\text{NewTravel}} = T_3^{\text{OldTravel}} + \alpha$$
   $$T_4^{\text{NewTravel}} = T_4^{\text{OldTravel}} + \alpha$$

   Thus since we are storing the old travel time in the queue, everytime we insert the data in the middle, we just need to add α to the following data in the queue, thus avoiding the need to calculate the R(t), every time.

6. **Video of the Implementation running**

   Zoom (link & password):
   https://ucd-ie.zoom.us/rec/share/WCVJN-MGMuDz2ZJmL_DQjekFGHxjVFrAwVCz_JRkQCn0SLKQztLx4O3h7_LE21fI.Ope2SQ5mUTwH8B6P?startTime=1677444449000
   Passcode:
   4AGYe%2$

7. **Comments:**
   The execution was slow because of low system configuration. Execution time varies on different systems.