Discrete II Project

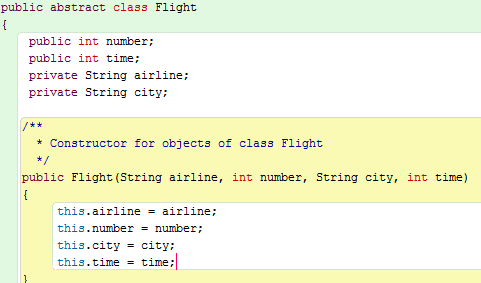
Airport Gate Assignment Problem

Justin Arends

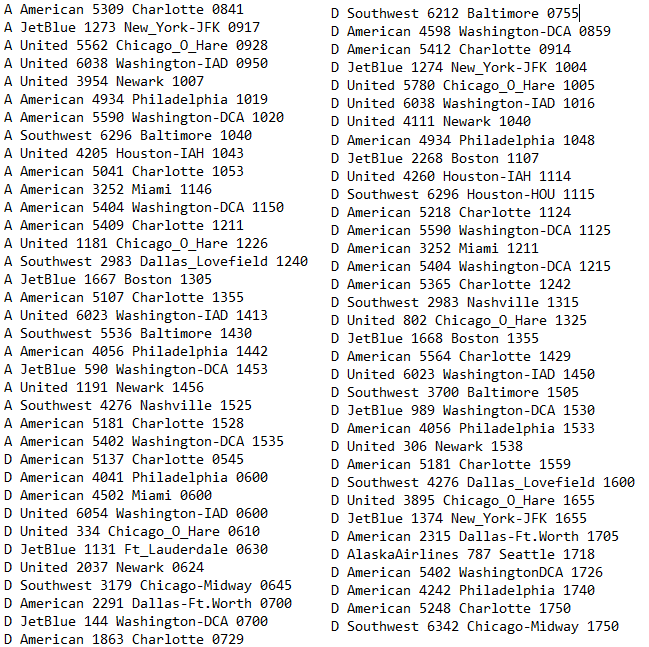
College of Charleston

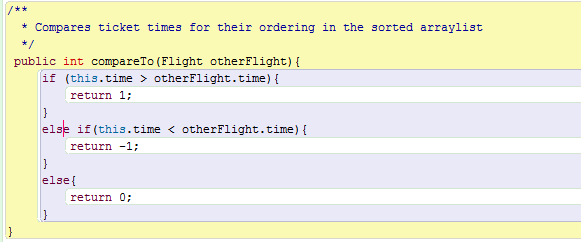
04/26/2017

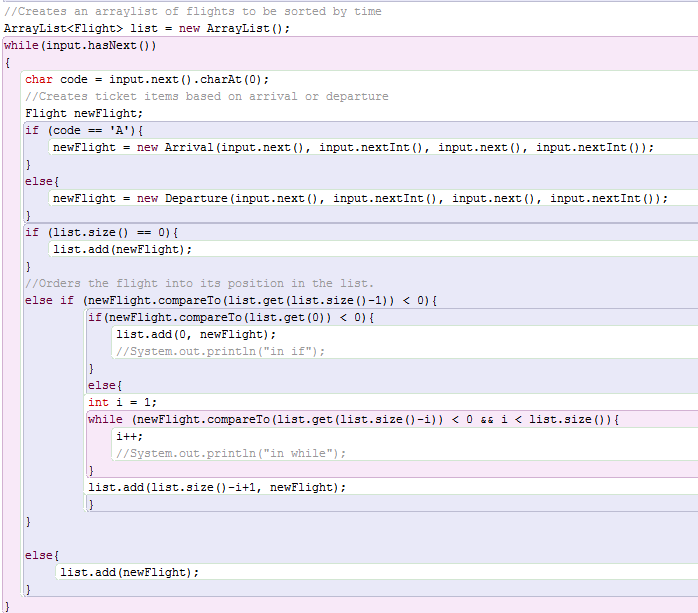
In this project I sought to combine discrete mathematical skills with java programming to solve the problem of scheduling arriving and departing aircraft to the least number of necessary gates as safely and efficiently as possible. To do this it should be noted that no more than one flight can occupy any gate at a given time, and also no flight can occupy more than any one gate. To further decrease the possibility of conflict between flights, I added a thirty minute lock period for which an aircraft claims a gate. I then constructed flight objects with subclasses for arrival and departure:

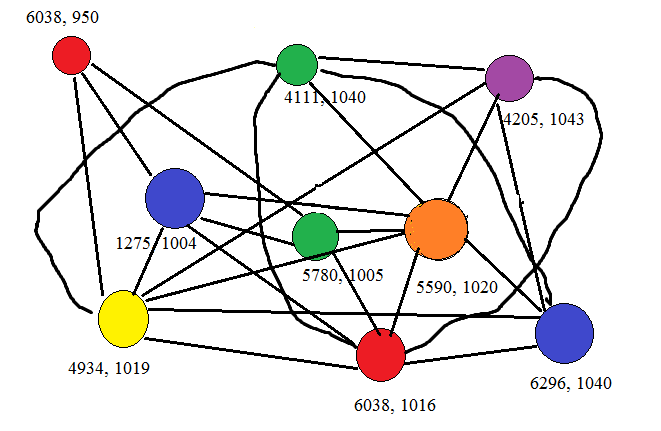


My first approach was to create smaller and more manageable problems, starting with collecting all of the data required. I selected a day of flights at the Charleston International Airport to test with, ignoring Delta flights as they have their own separate concourse. These flights (or any others) would be placed in a file to be read into the program to create an object for each flight. To begin to organize the data, these flight objects would be placed into an ArrayList and ordered by their time with a simple sorting algorithm. The program will then print the contents of the ArrayList to display every flight in order.

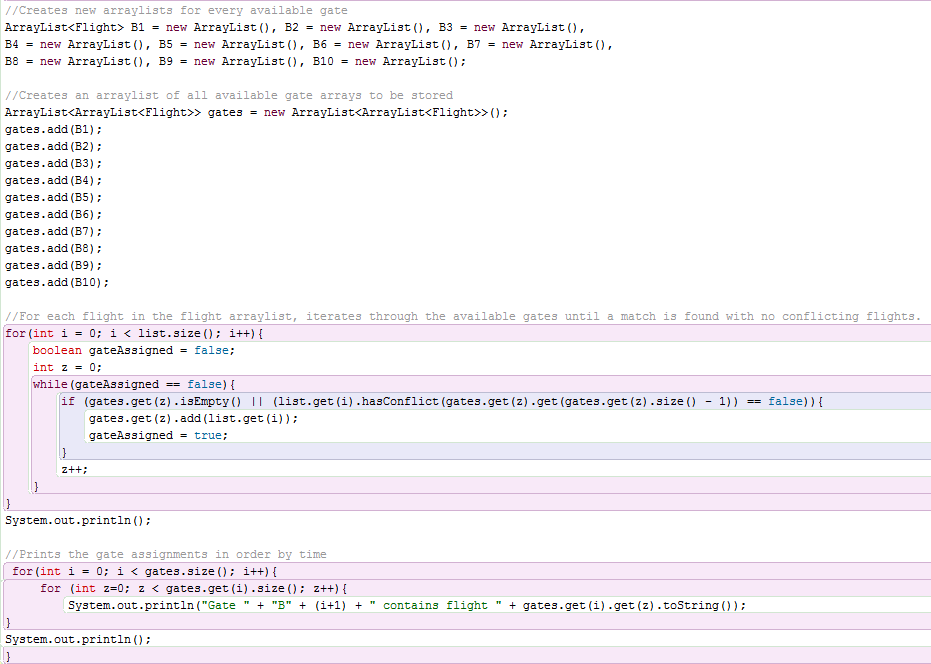
This is the list of flights I used with “A” representing arrival and “D” representing a departure. The time is displayed in 24 hour format to the right of each flight. It would be read in from an input file via the scanner:

****Below shows how flight objects are created and sorted into the ArrayList of flights needing to be assigned a gate. It places both arrivals and departures in the ArrayList with the distinction of two different types of flight objects:

****

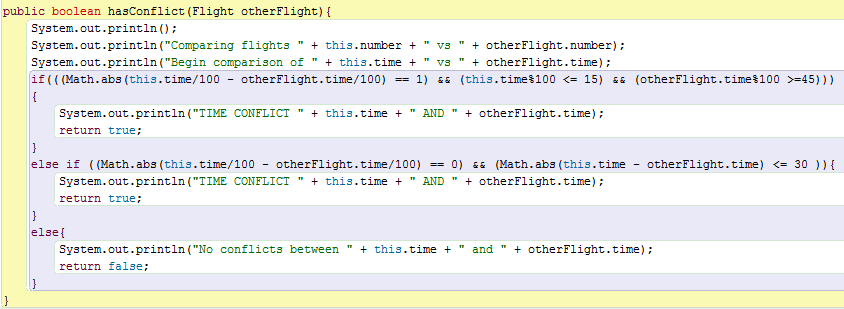
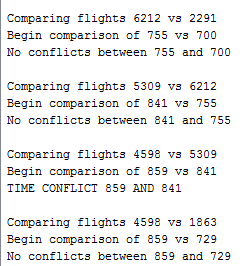
Obviously a visual model of this problem would be large and messy, and would be difficult to make sense of if drawn by the computer. This is a small sample mapping of a one hour time block to demonstrate how gate assignment may look:

To further break apart the problem, I created an ArrayList for every gate, and their own ArrayList to be stored in. This has the use of allowing to iterate through available gates in order to appropriately match a flight to them. I use a nested loop as I go down the list of flights to be assigned in order to also loop through available gates, prioritizing the gates closer to the entrance. A flight is automatically assigned to a gate if it has not yet been assigned any aircraft. Otherwise, a method is called to determine if the aircraft to be assigned conflicts with any of the aircraft currently assigned to that gate. If so, the loop moves on to the next gate and repeats this process until a gate can be assigned to the aircraft with no conflicts.

The code below shows how I created a container for each individual gate to hold flights that are assigned, and another container to hold all of the gates to be checked for assignment:

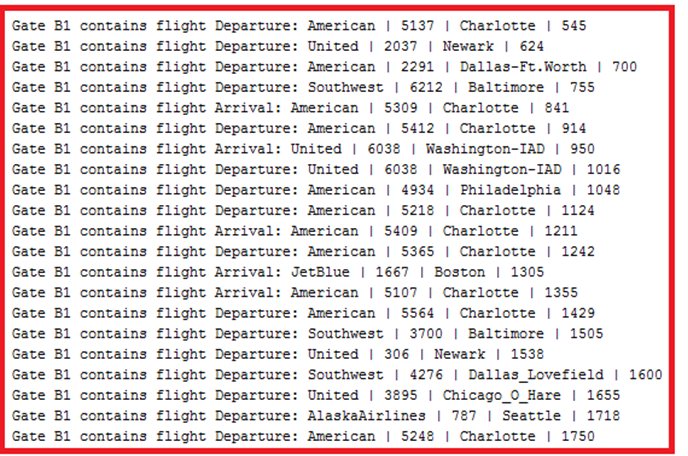
The hasConflict method will return a conflict if the aircraft to be assigned in any way clashes with the lockdown period of an aircraft that has already been assigned, and causes the program to move on to the next gate. If however no conflicts arise, the aircraft will be assigned to the gate of highest priority and the outside loop will move on to the next flight to be assigned.

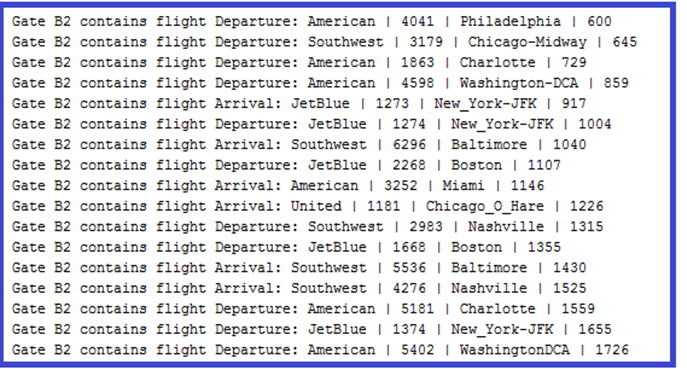
Below is my best attempt at creating a method that will check if two flights fall within eachother’s time slot, also taking a fifteen minute buffer time before and after their assigned time. Considerations were taken to work integers in a fashion that minutes could be added and subtracted and return the correct time:



The output information to the right is an example of what will be returned by the hasConflict method. It will display the current flight in the arraylist to be sorted vs the flight in the current gate arraylist to be checked, their assigned time, and whether or not there is a conflict. It will continue comparing until the flight is assigned, before moving on to the next flight.

Ignoring the issue of giving priority to roundtrip flights for them to be able to reliably use the same gate for arrival and departure, I was able to use only six of the ten available gates for the given day with no unsafe conflicts and no needlessly wasted time or resource allocation.

The results depicted below are sorted first by gate, then the flight’s arrival or departure time on the right of the line. By the nature of this program, the lower the number of the gate, the higher priority. In reality, the priority of these gates makes no difference other than lower numbers being closer to the airport entrance:





There are clearly many and more advanced methods to solve this problem that factor in many more real world considerations, however this approach seems feasible for basic scheduling problem, especially those with excessively large amounts of data.

I am certain that with a greater knowledge of java data structures, sorting algorithms, comparator methods, and time utilities I could further increase the accuracy of the output as well as greatly increase the efficiency by reducing the time complexity. However I believe the results proved to be satisfactory in achieving the problem I set out to solve, and thus I have shown that a day’s worth of flights at Charleston International Airport could be reduced to only requiring a minimum of six gates.