Sabanci University

Faculty of engineering and Natural Sciences

CS 300 Data Structures

Assigned: December 17, 2020 Due: Dec 28, 2020 @ 11:55pm

Please note:

SOLUTIONS HAVE TO BE YOUR OWN. NO COLLABORATION OR COOPERATION AMONG STUDENTS IS PERMITTED.

10% PENALTY WILL BE INCURRED FOR EACH DAY OF OVERTIME. SUBMISSIONS THAT ARE LATE MORE THAN 3 DAYS WILL NOT GET ANY CREDITS.

SUBMISSIONS WILL BE MADE TO THE SUCOURSE SERVER. NO OTHER METHOD OF SUBMISSION WILL BE ACCEPTED.

Introduction

Abstract data structures (ADS) are some of the most important concepts you will need in your journey as a developer. They are so important, in fact, that most languages come already equipped with implementations of most of the standard ADS like hashtables, priority queues, stacks, queues, and linked lists. This includes everyone's favorite language C++.

The C++ standard library (the part of the language that includes things like `cout` and the `vector` class) includes implementations to many useful data structures. In this assignment, you will solve a programming problem that will require the usage of one or more of these ADSs. We will provide a small cheat sheet to help you get started with two standard library classes that we think will help you solve this problem. You can find this document uploaded to SUCourse+ along with this document.

You might be wondering, "if all of these data structures already exist in C++, then why do we need to learn how to implement them?" First of all, understanding how the implementation is done and doing it yourself gives you a much better understanding of why these data structures have their respective complexities, what situations they are useful for, what design decisions were made and why they were made Also, in many instances, you might need to make small modifications on the data structures to suit your specific use case (think of last week's homework. You had to implement your *own* modified priority queue.) Unless you understand the data structures it would be very difficult to make any such modifications.

In this assignment, you will design a booking system for a sports stadium. For various technical reasons, every function that you need to implement for this assignment **must abide by the time complexity constraints listed with it.** Failing to do so will lead to penalties in your grades. Please be mindful of the data structures you use and the complexities of your functions.
The stadium is split into **blocks** of seats. Each block has a name and is made up of named **rows** and numbered **columns**. An example of a stadium with the three blocks, each with four columns and six rows is shown below. The naming convention of a seat is as follows: `BlockName RowName-ColumnNum`. Figure 1 shows an example of a stadium and the naming of some of its seats.

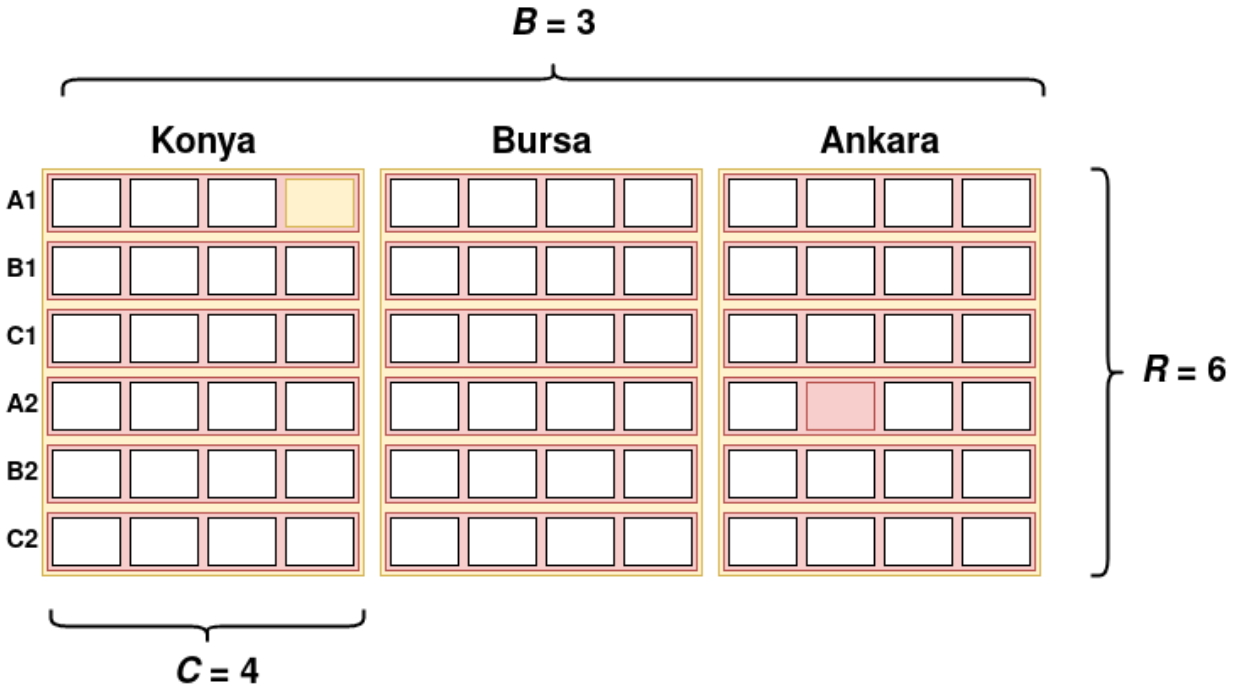

Figure 1: A stadium with three blocks named Konya, Bursa, and Ankara.. Each block is made up six rows named A1, B1, C1, A2, B2, C2. Every row has three columns. The seat highlighted in yellow has the name Konya A1-3. The seat highlighted in red has the name Ankara A2-1.

You must implement functions to reserve seats for customers in the stadium as well as cancel reservations. In addition, you will write functions to query particular customers to check if they have a reservation or not, and to check what the seat of a customer is (in case a customer has made a reservation.) Note that every customer can make at most one reservation. Furthermore, there will be two types of reservations. The first will select a particular row, column, and block to reserve. The second will select a row only, and the reservation must be made **in the block, which has the least number of reservations for the given row.** You will implement these functions with certain time complexity constraints. Your code will process queries from an input text file and output results to an output text file.

One final piece of advice: ***test your code incrementally as you write it***. The standard library makes so many things a lot easier to do, but it also can prove to be very hard to debug. So, make sure you test your code every few lines. Here is a related quote from the creator of C++:
"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off."

-- Bjarne Stroustrup, creator of C++

Implementation

You will write a **main program** that will take a single text file called "*inputs.txt*" which will contain the specifications of the stadium (block names, row names, and number of columns), and then it will have a list of commands, each in a line. Your program will create a single stadium based on the specifications, then carry out the commands in the file. Any output from the commands must be printed into a file called "*output.txt*". A sample test run is shown at the end of this document.

Note: During your implementation, you don't have to implement any classes, but you are free to do so if you think it will be helpful.

The input file will start with three lines. The first line lists out the names of the blocks, separated by a space. The second line lists out the names of the rows, separated by a space, and the third line contains a single integer, which is the number of columns in a single row, i.e.:

```
<BlockName1> <BlockName2> <BlockName3> ... <BlockNameB>
<RowName1> <RowName2> <RowName3> ... <RowNameR>
<NumberOfColumns>
```

According to this input, you will create the stadium data structure. Please note the following constraints:

1) There will be no duplicate row names and no duplicate block names.
2) The number of blocks ***B*** >=2
3) The number of rows ***R*** >=2
4) The number of columns ***C*** >=2
5) There will be no special characters or tab (\t) characters in these lines; only spaces between names and end lines (\n) at the end of a line.

Also, with this input we will define the following three values:

B **=** the number of blocks
R **=** the number of rows per block
C **=** number of columns per block

We will use these values in the remainder of the document when describing time complexities requirements.

In the following subsections, we will explain the commands that will occur in the input text, and the operations that they must perform.

`print`

This command will print out the current stadium state. It will print the blocks one after another, in the order in which their names occurred in the input text file. First, the name of the block will be printed, then each row will be printed in the same order in which the row names occurred in the input text file. When printing a row, if a seat is taken by a customer, **print the first three letters of their name.** If the seat is empty, print '---'. Columns must be separated by a single space character. Here is the format of printing a single block. **There will be no \t characters in the output.**

```
<BlockName>
~~~~~~~
<Row1Name> : --- --- Abc ---
<Row2Name> : --- Def --- ---
...
<RowRName> : --- --- --- ---
=======
```

The time complexity of this operation should be O(B * R * C)

Please check the test run at the bottom of the document for another example.

`reserve_seat_by_row <customer_name> <row_name>`

This function will reserve a seat for the customer `"customer_name"` in the row `"row_name"`. However, it will **find the block, in which the request row has the least number of reservations, and assign the leftmost empty seat in that row**. You may assume that `"row_name"` will always be one of the given row names at the beginning of the input text file, so you don't need to do any input checks. The following diagram shows an example of a reservation operation:

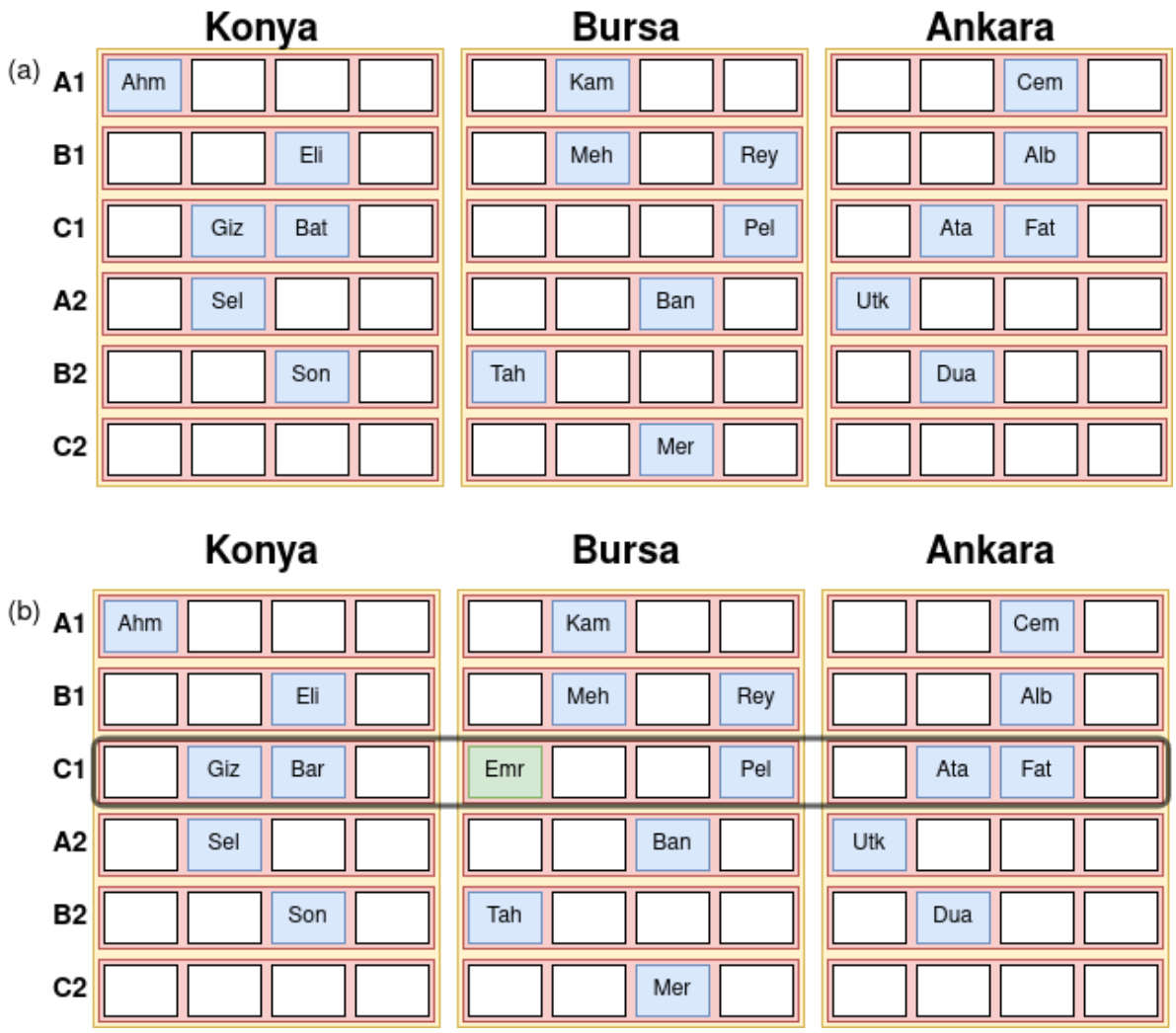

Figure 2: an example of a reserve_seat_by_row operation. (a) The stadium has three blocks; Konya, Bursa, and Ankara. Each block has six rows named A1, B1, C1, A2, B2, and C2, and four columns per row. The white seats are empty, and the blue ones are reserved. (b) when making the following command:

```
reserve_seat_by_row Emre C1
```

the block in which row `C1` is the emptiest is chosen, i.e block `Bursa`. We assign the seat in the leftmost empty column of that row, i.e column 0, to the customer.

After the customer is assigned a seat, the following output will be printed in the output.txt file:

```
<customer_name> has reserved <block_name> <row_name>-<column_number> by
emptiest block
```

If an empty seat isn't found, or it the customer already has a reserved seat, print the following line to the output text file:

```
<customre_name> could not reserve a seat!
```

Please note that in the case of two more blocks having the same number of empty seats in row `"row_name"`, *select the row in the block that occurs first in the first line of the input file.*

The time complexity of this operation should be O(Log*B* + *C*)

`reserve_seat <customer_name> <block_name> <row_name> <column_number>`

Where `customer_name`, `block_name`, and `row_name` are strings, while `column_number` is an integer.

This function will reserve the seat in the block named `"block_name"`, in the row `"row_name,"` and at the column `"column_number"` for the customer named `"customer_name"`. You may assume that `"block_name"` is the name of one of the blocks given at the beginning of the input text file, `"row_name"` is a name of one of the rows given at the beginning of the input text file, and `"column_number"` < ***C*** (number of columns)**.** You don't have to do any input checks for these three inputs. However, the customer `"customer_name"` could already have a reservation, in which case the reservation process will fail. This is because a single person can reserve a single seat only.

If the seat is reserved successfully, print out to the output text file:

```
<customer_name> has reserved <block_name> <row_name>-<column_number>
```

If the reservation fails, either because the seat is full or because the customer already has a reservation, print out:

```
<customre_name> could not reserve a seat!
```

The time complexity of this operation should be O(LogB)

Note: you might be wondering why the time complexity for this command is O(LogB) even though it can be implemented in O(1). The reason is that *the data structures required to satisfy the time constraints on the previous command (reserve_seat_by_row) lead to degrading this function's performance.*

`get_seat <customer_name>`

Print out to the output text file the reserved seat of the customer `"customre_name"` in the following format:

```
Found that <customre_name> has a reservation in <block_name>
<row_name>-<column_number>
```

If a customer with the name "customer_name" does not have a reservation, print out to the output text file:

```
There is no reservation made for <customre_name>!
```

The time complexity of this operation should be O(1)

`cancel_reservation <customer_name>`

If a customer with the name "customer_name" has a reservation in the stadium, cancels their reservation, makes their seat, and prints to the output file:

```
Cancelled the reservation of <customre_name>
```

If the customer "customre_name" doesn't have a reservation, prints to the output file:

```
Could not cancel the reservation for <customre_name>; no reservation found!
```

Please note that after a customer cancels their reservation, they are allowed to make a new reservation.

The time complexity of this operation should be O(*LogB*)

Sample Run

Please find below an example input.txt file, followed by the expected output.txt file.

inputs.txt

```
Konya Bursa Ankara
A1 B1 C1 A2 B2 C2
4
print
reserve_seat Ahmet Konya A1 0
reserve_seat Kamer Bursa A1 1
reserve_seat Cemal Ankara A1 2
reserve_seat Elif Konya B1 2
reserve_seat Mehmet Bursa B2 1
reserve_seat Reyyan Bursa B1 3
```

```
print
reserve_seat Albert Ankara B1 2
reserve_seat Gizem Konya C1 1
reserve_seat Batuhan Konya C1 2
reserve_seat Pelinsu Bursa C1 3
reserve_seat Atahan Ankara C1 1
print
reserve_seat Fatih Ankara C1 2
reserve_seat Selim Konya A2 1
reserve_seat Banu Bursa A2 2
cancel_reservation Utku
reserve_seat Utku Ankara A2 0
cancel_reservation Utku
reserve_seat Soner Konya B2 2
reserve_seat Taha Bursa B2 0
get_seat Dua
reserve_seat Dua Ankara B2 1
get_seat Dua
reserve_seat Meriam Bursa C2 2
print
reserve_seat_by_row Emre C1
print
```

outputs.txt

```
Konya
~~~~~~~
A1 : --- --- --- ---
B1 : --- --- --- ---
C1 : --- --- --- ---
A2 : --- --- --- ---
B2 : --- --- --- ---
C2 : --- --- --- ---
=======

Bursa
~~~~~~~
A1 : --- --- --- ---
B1 : --- --- --- ---
C1 : --- --- --- ---
A2 : --- --- --- ---
```

```
B2 : --- --- --- ---
C2 : --- --- --- ---
=======

Ankara
~~~~~~~
A1 : --- --- --- ---
B1 : --- --- --- ---
C1 : --- --- --- ---
A2 : --- --- --- ---
B2 : --- --- --- ---
C2 : --- --- --- ---
=======

Ahmet has reserved Konya A1-0
Kamer has reserved Bursa A1-1
Cemal has reserved Ankara A1-2
Elif has reserved Konya B1-2
Mehmet has reserved Bursa B2-1
Reyyan has reserved Bursa B1-3

Konya
~~~~~~~
A1 : Ahm --- --- ---
B1 : --- --- Eli ---
C1 : --- --- --- ---
A2 : --- --- --- ---
B2 : --- --- --- ---
C2 : --- --- --- ---
=======

Bursa
~~~~~~~
A1 : --- Kam --- ---
B1 : --- --- --- Rey
C1 : --- --- --- ---
A2 : --- --- --- ---
B2 : --- Meh --- ---
C2 : --- --- --- ---
=======

Ankara
~~~~~~~
```

```
A1 : --- --- Cem ---
B1 : --- --- --- ---
C1 : --- --- --- ---
A2 : --- --- --- ---
B2 : --- --- --- ---
C2 : --- --- --- ---
=======

Albert has reserved Ankara B1-2
Gizem has reserved Konya C1-1
Batuhan has reserved Konya C1-2
Pelinsu has reserved Bursa C1-3
Atahan has reserved Ankara C1-1

Konya
~~~~~~~
A1 : Ahm --- --- ---
B1 : --- --- Eli ---
C1 : --- Giz Bat ---
A2 : --- --- --- ---
B2 : --- --- --- ---
C2 : --- --- --- ---
=======

Bursa
~~~~~~~
A1 : --- Kam --- ---
B1 : --- --- --- Rey
C1 : --- --- --- Pel
A2 : --- --- --- ---
B2 : --- Meh --- ---
C2 : --- --- --- ---
=======

Ankara
~~~~~~~
A1 : --- --- Cem ---
B1 : --- --- Alb ---
C1 : --- Ata --- ---
A2 : --- --- --- ---
B2 : --- --- --- ---
C2 : --- --- --- ---
=======
```

```
Fatih has reserved Ankara C1-2
Selim has reserved Konya A2-1
Banu has reserved Bursa A2-2
Could not cancel the reservation for Utku; no reservation found!
Utku has reserved Ankara A2-0
Cancelled the reservation of Utku
Soner has reserved Konya B2-2
Taha has reserved Bursa B2-0
There is no reservation made for Dua!
Dua has reserved Ankara B2-1
Found that Dua has a reservation in Ankara B2-1
Meriam has reserved Bursa C2-2

Konya
~~~~~~~
A1 : Ahm --- --- ---
B1 : --- --- Eli ---
C1 : --- Giz Bat ---
A2 : --- Sel --- ---
B2 : --- --- Son ---
C2 : --- --- --- ---
=======

Bursa
~~~~~~~
A1 : --- Kam --- ---
B1 : --- --- --- Rey
C1 : --- --- --- Pel
A2 : --- --- Ban ---
B2 : Tah Meh --- ---
C2 : --- --- Mer ---
=======

Ankara
~~~~~~~
A1 : --- --- Cem ---
B1 : --- --- Alb ---
C1 : --- Ata Fat ---
A2 : --- --- --- ---
B2 : --- Dua --- ---
C2 : --- --- --- ---
=======
```

```
Emre has reserved Bursa C1-0 by emptiest block

Konya
~~~~~~~
A1 : Ahm --- --- ---
B1 : --- --- Eli ---
C1 : --- Giz Bat ---
A2 : --- Sel --- ---
B2 : --- --- Son ---
C2 : --- --- --- ---
=======

Bursa
~~~~~~~
A1 : --- Kam --- ---
B1 : --- --- --- Rey
C1 : Emr --- --- Pel
A2 : --- --- Ban ---
B2 : Tah Meh --- ---
C2 : --- --- Mer ---
=======

Ankara
~~~~~~~
A1 : --- --- Cem ---
B1 : --- --- Alb ---
C1 : --- Ata Fat ---
A2 : --- --- --- ---
B2 : --- Dua --- ---
C2 : --- --- --- ---
=======
```