

Project 2

- Choose either project 2A, 2B, or 2C. **Deadline: November 11, 2016.** All the submissions should be made electronically on Blackboard by the end of the day.
- Please stick to the teams that I assigned you with.

Project 2B

- Using *queues*, design a system that allows the circulation of books to employees.

Technical Requirements

- (Weight: 10%) The system should keep track of two lists: (1) Books to be circulated: these are the books that will be circulated to employees. (2) Archived books: when the last employee on the queue returns the book to the library, the book gets added to the list of archived books.
- (Weight: 5%) The system should keep track of employees: these are the employees that will get the books later on.
- (Weight: 25%) Circulate a book to the employees in the system. The circulation starts on a given date. Further, the system makes a queue of employees that should receive the book. The queue should be prioritized based on two factors: (1) the total waiting time for the employee: How many days the employee waited to get a book since the beginning of the circulation. (2) The total retaining time: How many days the employee retained books.
- (Weight: 20%) Make a data structure (priority queue) that allows the pushing and popping of items. The popped item is the item with the highest priority. The queue should also be updatable whenever an item's priority changes.
- The more the employee waited, the higher the priority. The more she retained a book, the lower the priority. To put it simply, the priority is: *waiting_time – retaining_time*. The employee in front of the queue gets the book.
- (Weight: 40%) The system should allow the employee to pass the book on to the next employee on the queue on a given date. Use parallel programming to speed up the performance. Passing on the book has the following outcome:
 - ✓ If the employee who is passing on the book is the last in the queue, the book gets archived.
 - ✓ The total retaining time for the employee who passed on the book gets adjusted.
 - ✓ The total waiting time for the employee who got the book gets adjusted.
 - ✓ If there are other queues for other books, and these queues contain the employee who passed on the book and the employee who got the book, then adjust these queues (because the priorities have changed).
 - ✓ See Figure 1 for an illustration.



Waiting line for the Software Engineering book

Ann
Sam
Adam

Queue

After 3 days



Ann passes the book on to Sam

Employee	Retaining time	Waiting time
Ann	0	0
Sam	0	0
Adam	0	0

Sam
Adam

Queue

After 1 day



Sam passes the book on to Adam

Employee	Retaining time	Waiting time
Ann	3	0
Sam	0	3
Adam	0	0

Adam

Queue

After 5 days



Sam passes the book on to the library so that they archive it

Employee	Retaining time	Waiting time
Ann	3	0
Sam	1	3
Adam	0	4

Queue is empty, and the book is archived

Queue

Employee	Retaining time	Waiting time
Ann	3	0
Sam	1	3
Adam	5	4

Figure 1: An illustration of circulating the software engineering book.

Facts and Assumptions

- A book has the following properties: name, circulation start date, circulation end date, archived (whether the book is archived), a queue of employees (the employees who are planned to get the book).
- An employee has the following properties: name, waiting time, and retaining time.
- There is only one copy of each book the system has.
- You can use any source code on Blackboard or any built-in c++ data structures (apart from priority queue).
- You can use the `Date` class I wrote (it has year, month, day components). You can find a project that shows examples of how to use the `Date` class here:
<https://www.dropbox.com/sh/1bfc5ayty5kl16x/AAD-bO28NQX7RaFQj39ZRQV1a?dl=0>
- There is no need for getting input from the user or a menu-based system. For instance, this is an example of how the circulation system (e.g. library) is used in the main function:

```
int main(){
    Library library;
    library.add_book("Software Engineering");
    library.add_book("Chemistry");

    library.add_employee("Adam");
    library.add_employee("Sam");
    library.add_employee("Ann");

    library.circulate_book("Chemistry", Date(2015, 3, 1, DateFormat::US));
    library.circulate_book("Software Engineering", Date(2015, 4, 1, DateFormat::US));
    library.pass_on("Chemistry", Date(2015, 3, 5, DateFormat::US)); //tell the next employee
to pass the book on March 5, 2015
    library.pass_on("Chemistry", Date(2015, 3, 7, DateFormat::US));
    library.pass_on("Chemistry", Date(2015, 3, 15, DateFormat::US)); //at this point in time,
the system will archive the chemistry book.

    library.pass_on("Software Engineering", Date(2015, 4, 5, DateFormat::US));
    library.pass_on("Software Engineering", Date(2015, 4, 10, DateFormat::US));
    library.pass_on("Software Engineering", Date(2015, 4, 15, DateFormat::US));
}
```

Project 2B (Open Project)

- Choose some of the data structures we studied or make your own to help a bookstore give customers recommendations for books they may be interested in. The system will also help the customer find books efficiently.
- Here is how the system works: The customer logs into the system (simply by entering his/her ID). The system gives the customer following options: (1) Search for a book and probably rate it. (2) Rate a book they had not rated before. (3) View book recommendations sorted by relevance.
- Here is a suggestion for making recommendations for customers: There are two algorithms: Algorithm A and algorithm B:

Algorithm A

If customer has rated some books {

 If you find customers that rated some of the same books {

- Create a weight for each customer that reflects how close their rating is to yours.
- To calculate the weight you need to take into consideration: number of books you and the customers rated and difference of rating. The greater the difference, the lower the weight. The more the common books, the higher the weight.
- Create a list of recommendations based on algorithm B (see below)

 }

 else{

- the recommended books will be the most popular books (the books with the highest ratings) }

 }

 }

else {

- the recommended books will be the most popular books (the books with the highest ratings) }

Algorithm B ~ making recommendations based on a list of similar customers

 Iterate through the list of similar customers (customers who like the same books)

 Iterate through the list of books they rated

 When you find a book they gave a high rating (and you didn't rate at all), add it to the list of recommendations

Technical Requirements

- (Weight: 20%) Use data structures that store books/customers/and ratings. You can find sample data here
<https://www.dropbox.com/sh/w260nkp2ergfuno/AADICsulRt6pCYQ58qveHYD8a?dl=0>

- (Weight: 20%) Write an efficient function that finds books for customers. The customer may find a book by ISBN or book title (they don't have to enter the entire name or ISBN). Use parallel programming to speed up the function.
- (Weight: 60%) Write an efficient function that makes book recommendations for books. The recommendations should be sorted by relevance. Use parallel programming to speed up the algorithm.

Discussion

- Since this is an open project, the grading will largely depend on the thinking process and design decisions. Hence, you are welcome to discuss with me how you will approach the problem. Please note that the algorithms I highlighted above are only suggestions. Feel free to deviate from them.

Project 2C (Open Project)

- Choose some of the data structures we studied or make your own to simulate how an elevator handles and keeps track of requests made by users.
- Here is how an elevator works: A user pushes either the up or down button to request the elevator. Once the elevator arrives, the user gets in and specifies his/her destination floor. Meanwhile, other users outside or inside the elevator make requests as well. The elevator stores the requests and handles them efficiently.
- A simple algorithm for handling requests works like this: all requests users make are stored. The elevator prioritizes the requests that are on the way where it's going, but also based on a first come first served principle. It is up to you how you design the algorithm. As a recommendation, observe how an elevator works (e.g. the elevator we have at FH) and simulate how it works.

Technical Requirements

- (Weight: 60%) Write an efficient function that handles elevator requests. The function should result in a low waiting time for users. Make sure your function so that accommodates a single-elevator system or a multi-elevator system. Use parallel programming to speed up the performance.
- (Weight: 40%) Write a simulation function that calculates the waiting time for users. Your simulation algorithm could randomly generate users that come at random times and make random requests. Alternatively, make your own scenarios and test the waiting time for those scenarios. For inspiration, see how we simulated the airline passenger serving algorithm.

Discussion

- Since this is an open project, the grading will largely depend on the thinking process and design decisions. Hence, you are welcome to discuss with me how you will approach the problem.