Fundamentals of Engineering Algorithms – Fall 2024

Concert Ticket Reservation System

Ester Vicente
Taina Nieves

</>

Introduction

01

} /> [

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

Goals and Project Scope

</ Introduction

Our idea:

- To build a system to book concert tickets.
- The system would allow users to:
  - book or cancel seats
  - display all the available seats.
- The system would able to model a venue with multiple arrangements

Goals:

- To apply our knowledge of linked list and its techniques in order to create something simple and accessible for users.
- Use sorting techniques to make the system customizable to each user, allowing them to display the information in the order that is most convenient for them.

\\ Literature Review

- Mini project Assignment.
  - Created a flight booking system using Singly Linked Lists.
  - Used the booking, cancellation, and viewing available seats functions as a base.

- Wanted to expand on that idea with another system.
  - Ours include doubly linked list and different functions.
  - Shows all the options in the command window.

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</>

Methodology

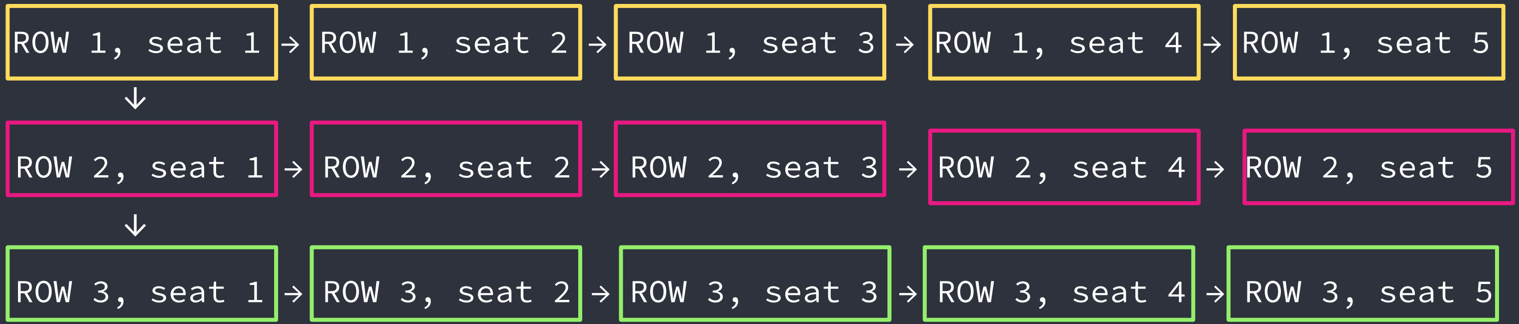
02

} /> [

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Methodology

Data Structure: Multilevel Linked List



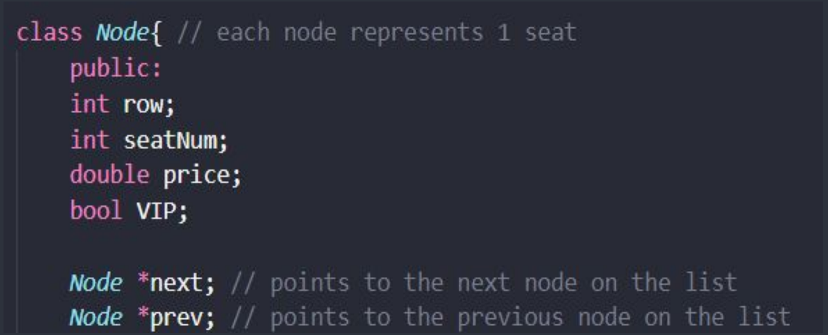
ROW 1, seat 1 → ROW 1, seat 2 → ROW 1, seat 3 → ROW 1, seat 4 → ROW 1, seat 5
↓
ROW 2, seat 1 → ROW 2, seat 2 → ROW 2, seat 3 → ROW 2, seat 4 → ROW 2, seat 5
↓
ROW 3, seat 1 → ROW 3, seat 2 → ROW 3, seat 3 → ROW 3, seat 4 → ROW 3, seat 5

</ Methodology

Classes used:

- Each Node represents a seat
- Each Row is a doubly linked list
- The Booking System class uses previous classes to generate the seating map and modify it

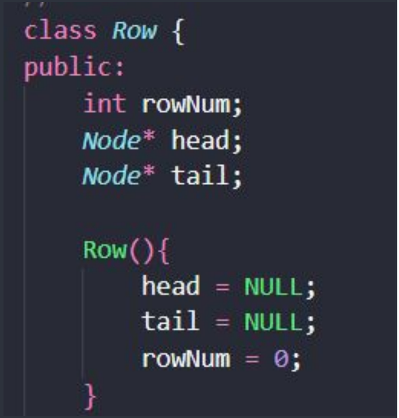
Seat (Node)



```
class Node{ // each node represents 1 seat
    public:
    int row;
    int seatNum;
    double price;
    bool VIP;

    Node *next; // points to the next node on the list
    Node *prev; // points to the previous node on the list
```

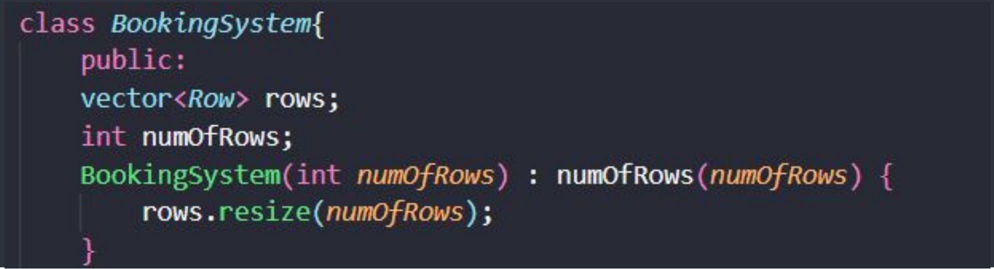
Row



```
class Row {
public:
    int rowNum;
    Node* head;
    Node* tail;

    Row(){
        head = NULL;
        tail = NULL;
        rowNum = 0;
    }
```

Booking System



```
class BookingSystem{
    public:
    vector<Row> rows;
    int numOfRows;
    BookingSystem(int numOfRows) : numOfRows(numOfRows) {
        rows.resize(numOfRows);
    }
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Methodology

Techniques:

Row Class

- Inserting node at the end of a linked list → *addSeat()*
- Removing a specific node in a linked list → *removeSeat()*
- Sorting based on price or seat number (quick sort) → *sortRowPrice()* and *sortRowSeatNum()*
- Initialize seats in row → *initRowSeats()*

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Methodology

Pseudocode for Techniques:

- Inserting node at the end of a linked list

```
If (head == NULL){ //there are no elements in the list
     add newNode
     head = newNode
     tail = newNode}
Else{
     tail -> next = newNode //inserts newNode after last tail pointer
     newNode -> prev = tail //links newNode prev pointer to current tail
     tail = newNode //updates tail pointer
}
```

</ Methodology

Pseudocode for Techniques:

- Removing a specific node in a linked list. Input = seatKey

```
If (head == NULL){ //there are no elements in the list
     “There are no objects to remove” }

Create temp pointer

While (temp != NULL and temp->seatNumber != seatKey){
     temp = temp -> next} //while we dont find the specific key transverse the list

If (temp == NULL){seat number does not exist}
If (temp -> next == NULL){temp ->prev ->next = temp->next} //update next pointer
If (temp -> prev == NULL){temp ->next ->prev = temp->prev} //update prev pointer
If (temp is head OR tail) {head = temp -> next or tail = temp -> prev} // update   head or tail

delete temp
```

</ Methodology

Pseudocode for Techniques:

- Quick sort:

```
Quick-Sort(A, left, right)
    if (left ≥ right return)
    else{
        middle ← Partition(A, left, right)
        Quick-Sort(A, left, middle-1 )
        Quick-Sort(A, middle+1, right)}
    end if
```

</ Methodology

Techniques:

Booking System

- Create rows with a determined number of seats → *createRows()*
- Remove specific seat number from a specified row → *removeRowSeat()*
- Inserting a seat in its sorted position (doubly linked list) → *insertSortedSeat()*
- Printing all existing nodes in a multilevel linked list → *printRowSeats()*

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1

</ Time Complexity

Functions:

- *addSeat* → O(1) linear function
- *removeSeat* → O(n)
- *sortRowPrice* → O(n log n) because of quick sort
- *sortRowSeatNum* → O(n log n) because of quick sort
- *initRowSeats* → O(n) where n is the number of seats
- *createRows* → $O(n^2)$
- *printRowSeats* → O(n)
- *insertSortedSeat* → O(n)
- *removeRowSeat* → O(1) linear function

Time complexity:

Worst case: $O(n^2)$

Average case: $\Theta(n)$

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</>

Analysis and Results

03

}/>[

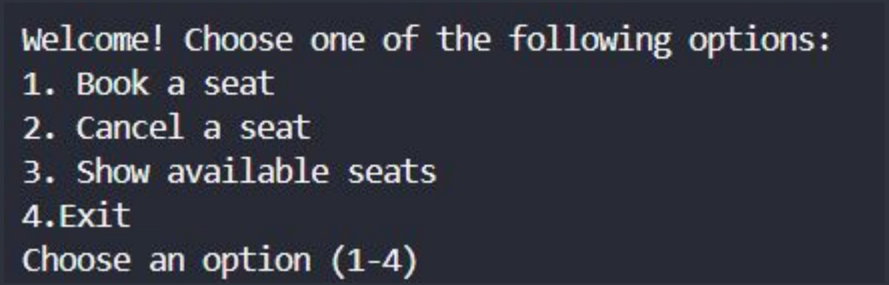
1011 011 01 1011001 10 11011 011 01 110110 110111 1101

<// Analysis and Results: Time Spent

- Time spend per week: 4 hours
- Time spent per month: 16 hours

</// Analysis and Results

User menu:

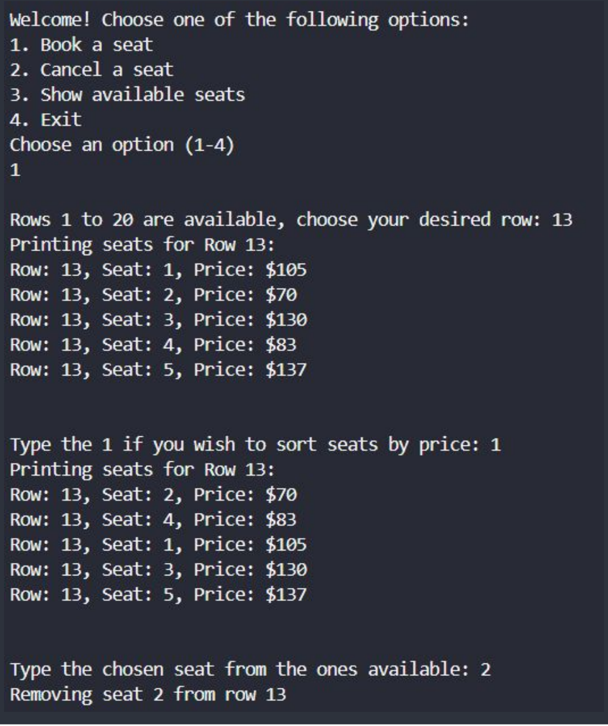


```
Welcome! Choose one of the following options:
1. Book a seat
2. Cancel a seat
3. Show available seats
4.Exit
Choose an option (1-4)
```

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</// Analysis and Results

Booking a seat:



```
Welcome! Choose one of the following options:
1. Book a seat
2. Cancel a seat
3. Show available seats
4. Exit
Choose an option (1-4)
1

Rows 1 to 20 are available, choose your desired row: 13
Printing seats for Row 13:
Row: 13, Seat: 1, Price: $105
Row: 13, Seat: 2, Price: $70
Row: 13, Seat: 3, Price: $130
Row: 13, Seat: 4, Price: $83
Row: 13, Seat: 5, Price: $137


Type the 1 if you wish to sort seats by price: 1
Printing seats for Row 13:
Row: 13, Seat: 2, Price: $70
Row: 13, Seat: 4, Price: $83
Row: 13, Seat: 1, Price: $105
Row: 13, Seat: 3, Price: $130
Row: 13, Seat: 5, Price: $137


Type the chosen seat from the ones available: 2
Removing seat 2 from row 13
```

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</>

Discussion

04

}/> [

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

% Discussion: Findings and Limitations

Findings:

- Advantage of using Doubly Linked Lists
  - Easily traverse forward and backward when canceling a seat and adding it back to the list and sorting.

Limitations:

- Not able to view map
- No advanced seating option such as VIP
- Little flexibility on adjusting to seat maps

</>

Conclusion

05

}/>[

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

/[Conclusion]

- Breaking the code down into classes helped keep the code neat.
- Using Double and Multilevel Linked Lists allowed for efficient insertion, deletion and traversal
  - Using Quick Sort with linked list was efficient.

For future research:

- Adding a user interface and a visualization of the seat map.
- Adding further division for the tickets, such as general admission, mezzanine etc, and make it customizable to each venue.
- Add VIP options

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

/[References]

- [1]“Multilevel Linked List,” *GeeksforGeeks*, Aug. 06, 2021. https://www.geeksforgeeks.org/multilevel-linked-list/