Assignment

Group: 07

Group Members:

Azizul Abedin Azmi	2022-1-60-130
Saifur Rahman	2021-3-60-033
Lamia Akter	2021-3-60-147
Mumtahina Akter	2021-3-60-137
Antara labiba	2021-3-60-173

Section: 04

Course Code: CSE246

Course Title: (Algorithms)

Date: 11/09/2024

Course Instructor:

Redwan Ahmed Rizvee

Adjunct Lecturer

Department of Computer Science and Engineering

1. Real-Life Problem Scenario: Airline Seat Assignment Problem

Problem Definition:

Imagine a budget airline that allows passengers to select their seats when purchasing a ticket. The airline wants to maximize the number of passengers seated while minimizing the number of adjacent empty seats to increase passenger comfort and operational efficiency. However, the airline has certain constraints, such as groups of passengers (friends or families) who wish to sit together, and business-class passengers who must be prioritized for better seats. The problem is to assign seats such that:

1. No adjacent seats remain empty when it's possible to avoid it.
2. Groups of passengers can sit together.
3. Business-class passengers get the best possible seats, which are closer to the front.

This problem can be modeled as a seat assignment problem, which can be solved using **Greedy Algorithms.** A greedy approach will allow for dynamically assigning the best seats to the passengers while respecting their preferences and minimizing gaps between seated passengers.

Constraints:

- Passengers can be grouped.
- No adjacent seats should remain empty unless unavoidable.
- Business-class passengers should be seated as close to the front as possible.

Input:

- A list of passengers with preferences (whether they are grouped, business-class, or general passengers).
- A seating arrangement represented by rows and columns in the plane.

Output:

- An optimized seat assignment where the conditions are satisfied.

2. Simulation: Airline Seat Assignment

Step-by-Step Solution Using a Greedy Algorithm:

Step 1: Sort passengers into two groups: business-class and general passengers. This is necessary because business-class passengers have a higher seating priority.

Step 2: Assign business-class passengers to the front seats, starting from the first available seat in the front row.

Step 3: Assign general passengers to the remaining seats, starting from the middle of the plane (to ensure gaps are minimized).

Step 4: Handle group seat assignments by checking adjacent seats and filling the group as close together as possible.

Example:

Let's say the plane has 3 rows with 4 seats in each row. The list of passengers includes:

- 3 business-class passengers
- 4 general passengers
- 2 groups of passengers (each group of 2 people)

The seating arrangement initially looks like this (B for business-class seat, G for general, and _ for empty):

```
Row 1: _ _ _ _
Row 2: _ _ _ _
Row 3: _ _ _ _
```

Step 1: Sort passengers:

- Business-class: B1, B2, B3
- General passengers: G1, G2, G3, G4
- Groups: Group1 (G5, G6), Group2 (G7, G8)

Step 2: Assign business-class passengers:

```
Row 1: B1 B2 B3 _
Row 2: _  _  _  _
Row 3: _  _  _  _
```

Step 3: Assign general passengers, starting from the middle:

```
Row 1: B1 B2 B3 _
Row 2: G1 G2 G3 G4
Row 3: _  _  _  _
```

Step 4: Assign group passengers while keeping them together:

```
Row 1: B1 B2 B3 _
Row 2: G1 G2 G3 G4
Row 3: G5 G6 G7 G8
```

3. Pseudo-code

```
CLASS Passenger:
    VARIABLES:
        - type: STRING  ("business" or "general")
        - groupSize: INTEGER (optional, default 0)

    CONSTRUCTOR (type, groupSize):
        SET type TO t
        SET groupSize TO g (default is 0)


FUNCTION assignSeats(seatingArrangement, passengers, startRow):
    SET rows TO number of rows in seatingArrangement
    SET cols TO number of columns in seatingArrangement
    SET index TO 0

    FOR each row from startRow to rows:
        FOR each column from 0 to cols:
            IF index is less than size of passengers AND seat is empty
(denoted by "_"):
                IF passenger type is "business":
                    ASSIGN seat to "B"
                ELSE:
                    ASSIGN seat to "G"
                INCREMENT index
            IF index is greater than or equal to size of passengers:
                BREAK out of loop


FUNCTION main:
    DEFINE seatingArrangement as 2D array of empty seats ("_")

    CREATE list of businessClass passengers
    CREATE list of general passengers

    CALL assignSeats with seatingArrangement, businessClass passengers, and
startRow 0 (front row)
    CALL assignSeats with seatingArrangement, general passengers, and
startRow 1 (middle and back rows)

    FOR each row in seatingArrangement:
        PRINT each seat in the row

    RETURN 0
```

4. C++ Implementation

```cpp
#include <iostream>
#include <vector>
using namespace std;

class Passenger {
public:
    string type;
    int groupSize;
    Passenger(string t, int g = 0) : type(t), groupSize(g) {}
};
void assignSeats(vector<vector<string>>& seatingArrangement,
vector<Passenger> passengers, int startRow) {
    int rows = seatingArrangement.size();
    int cols = seatingArrangement[0].size();
    int index = 0;
    for (int i = startRow; i < rows && index < passengers.size(); ++i) {
        for (int j = 0; j < cols && index < passengers.size(); ++j) {
            if (seatingArrangement[i][j] == "_") {
                seatingArrangement[i][j] = passengers[index].type ==
"business" ? "B" : "G";
                ++index;
            }
        }
    }
}
int main() {
    vector<vector<string>> seatingArrangement = {{"_", "_", "_", "_"},
                                                 {"_", "_", "_", "_"},
                                                 {"_", "_", "_", "_"}};

    vector<Passenger> businessClass = {Passenger("business"),
Passenger("business"), Passenger("business")};
    vector<Passenger> general = {Passenger("general"),
Passenger("general"), Passenger("general"), Passenger("general")};

    assignSeats(seatingArrangement, businessClass, 0);
    assignSeats(seatingArrangement, general, 1);
    for (const auto& row : seatingArrangement) {
        for (const auto& seat : row) {
            cout << seat << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output / Final Seating Arrangement:

Explanation of the output:

- **Row 0**: All seats are assigned to business-class passengers (B).
- **Row 1**: All seats are assigned to general passengers (G).
- **Row 2**: Remains empty with all seats as ("_") because there are no more passengers to assign.

When you run the program, this will be the displayed seating arrangement.

5. Complexity Analysis

Time Complexity:

1. **Initialization of Seating Arrangement**:
    - The seating arrangement is initialized as a 2D vector. Assuming the seating arrangement has rows and cols, initializing this matrix takes **O(rows * cols)** time.
2. **Assigning Business and General Class Passengers**:
    - The assignSeats() function is called twice:
        - First, for assigning business-class passengers starting from row 0.
        - Second, for assigning general passengers starting from row 1.

    In each call to assignSeats():

    - We iterate over the seating arrangement starting from a given row (startRow) and try to assign passengers to available seats ("_").
    - The outer loop runs for the number of rows (rows - startRow), and the inner loop runs for the number of columns (cols).
    - The index of passengers is incremented in each iteration until all passengers are seated or all seats are filled.

    Therefore, the time complexity of assigning seats for each class is proportional to the size of the seating matrix.

    Let n be the total number of passengers (business + general) and N = rows * cols be the total number of seats.

    The total time complexity for seat assignment is:

    - **O(min(n, N))** since the algorithm either assigns all passengers to available seats or stops when there are no more seats.

Summary:

- **Time Complexity**: **O(min(n, rows * cols))**