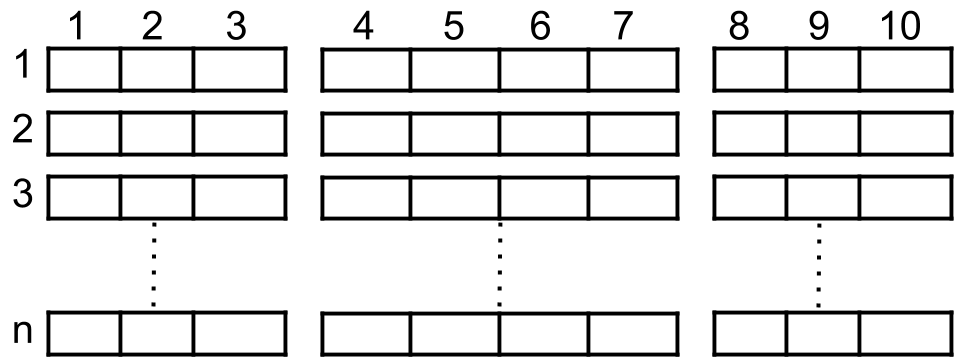
<https://leetcode.com/problems/cinema-seat-allocation/description/>

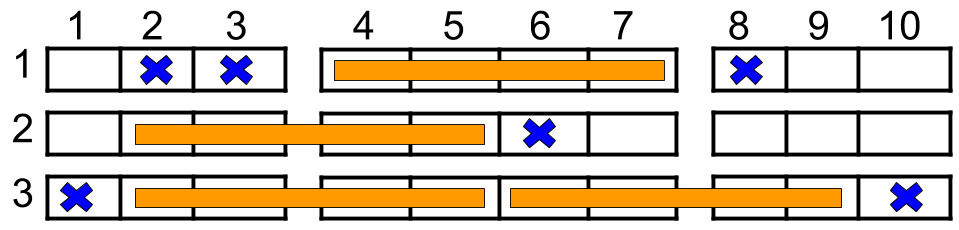


A cinema has n rows of seats, numbered from 1 to n and there are ten seats in each row, labelled from 1 to 10 as shown in the figure above.

Given the array reservedSeats containing the numbers of seats already reserved, for example, reservedSeats[i] = [3,8] means the seat located in row **3** and labelled with **8** is already reserved.

*Return the maximum number of four-person groups you can assign on the cinema seats.* A four-person group occupies four adjacent seats **in one single row**. Seats across an aisle (such as [3,3] and [3,4]) are not considered to be adjacent, but there is an exceptional case on which an aisle split a four-person group, in that case, the aisle split a four-person group in the middle, which means to have two people on each side.

**Example 1:**



**Input:** n = 3, reservedSeats = [[1,2],[1,3],[1,8],[2,6],[3,1],[3,10]]**Output:** 4**Explanation:** The figure above shows the optimal allocation for four groups, where seats mark with blue are already reserved and contiguous seats mark with orange are for one group.

**Example 2:**

**Input:** n = 2, reservedSeats = [[2,1],[1,8],[2,6]]**Output:** 2

**Example 3:**

**Input:** n = 4, reservedSeats = [[4,3],[1,4],[4,6],[1,7]]**Output:** 4

**Constraints:**

1 <= n <= 10^9

1 <= reservedSeats.length <= min(10\*n, 10^4)

reservedSeats[i].length == 2

1 <= reservedSeats[i][0] <= n

1 <= reservedSeats[i][1] <= 10

All reservedSeats[i] are distinct.

**Attempt 1: 2025-08-02**

**Solution 1: Hash Table (30 min)**

class Solution {

    public int maxNumberOfFamilies(int n, int[][] reservedSeats) {

        Map<Integer, Set<Integer>> map = new HashMap<>();

        for(int[] seat : reservedSeats) {

            int row = seat[0];

            int col = seat[1];

            map.putIfAbsent(row, new HashSet<>());

            map.get(row).add(col);

        }

        // To find maximum families

        // Rows with no reservations can fit 2 families

        int result = 2 \* (n - map.size());

        for(Set<Integer> reserved : map.values()) {

            // 1.First check if left and right are both available (no overlap, can place 2 families)

            // 2.Otherwise, check if any single block is available (left or middle or right)

            boolean left = !reserved.contains(2) && !reserved.contains(3)

                          && !reserved.contains(4) && !reserved.contains(5);

            boolean right = !reserved.contains(6) && !reserved.contains(7)

                           && !reserved.contains(8) && !reserved.contains(9);

            boolean middle = !reserved.contains(4) && !reserved.contains(5)

                          && !reserved.contains(6) && !reserved.contains(7);

            if(left && right) {

                result += 2;

            } else if(left || right || middle) {

                result += 1;

            }

        }

        return result;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

This problem involves allocating seats in a cinema hall where some seats are already reserved. The goal is to determine how many 4-person groups can be seated in the remaining available seats, following specific rules.

**Approach**

**Understand the seating arrangement**: The cinema hall has n rows with 10 seats each (labeled A-K, excluding I).

**Family seating rules**: A family of 4 must occupy 4 consecutive seats in the same row, either in B-E, D-G, or F-J.

**Reserved seats**: Some seats are already reserved, which affects where families can sit.

**Optimization**: For each row, check which of the 3 possible family seating blocks are available (not overlapping with reserved seats).

**Wrong Solution Code**

class Solution {

public int maxNumberOfFamilies(int n, int[][] reservedSeats) {

Map<Integer, Set<Integer>> reservedMap = new HashMap<>();

// Populate the reserved seats map

for (int[] seat : reservedSeats) {

int row = seat[0];

int col = seat[1];

reservedMap.putIfAbsent(row, new HashSet<>());

reservedMap.get(row).add(col);

}

int result = 2 \* (n - reservedMap.size()); // rows with no reservations can fit 2 families

for (Set<Integer> reserved : reservedMap.values()) {

boolean left = !reserved.contains(2) && !reserved.contains(3)

&& !reserved.contains(4) && !reserved.contains(5);

// Wrong logic here

boolean middle = !reserved.contains(6) && !reserved.contains(7)

&& !reserved.contains(8) && !reserved.contains(9);

boolean right = !reserved.contains(4) && !reserved.contains(5)

&& !reserved.contains(6) && !reserved.contains(7);

if (left && right) {

result += 2;

// Wrong logic here

} else if (left || middle || right) {

result += 1;

}

}

return result;

}

}

**Explanation**

**Initialization**: We first create a map to store reserved seats for each row.

**Processing Reserved Seats**: For each reserved seat, we add its column number to the corresponding row's set in the map.

**Calculating Maximum Families**:

Rows with no reservations can accommodate 2 families (left and right blocks), so we add 2 \* (total rows - reserved rows) to the result.

For each reserved row, we check the availability of the three possible blocks (left: B-E, middle: D-G, right: F-J).

If both left and right blocks are available, we can seat 2 families.

If any one of the three blocks is available, we can seat 1 family.

**Result**: The total count of families that can be seated is returned.

This approach efficiently checks the available seating blocks while minimizing the number of checks per row, leading to an optimal solution.

**Your solution is wrong with below input**

**Input**

**n = 2**

**reservedSeats = [[2,1],[1,8],[2,6]]**

**Output = 3**

**Expected = 2**

You're right. My previous solution incorrectly calculates the number of families for the given input. Let's analyze why and then provide a corrected solution.

**Analysis of the Issue**

For the input:

n = 2 (rows 1 and 2)

reservedSeats = [[2,1],[1,8],[2,6]]

Row 1 has seat 8 reserved, and row 2 has seats 1 and 6 reserved.

**Row 1:**

Reserved seats: {8}

Check blocks:

Left (B-E: 2-5): Available (no reserved seats)

Middle (D-G: 4-7): Seat 8 is not in this block, so available

Right (F-J: 6-9): Seat 8 is in this block, so not available

Can place at most 1 family (either left or middle)

**Row 2:**

Reserved seats: {1, 6}

Check blocks:

Left (B-E: 2-5): Available (no reserved seats)

Middle (D-G: 4-7): Seat 6 is in this block, so not available

Right (F-J: 6-9): Seat 6 is in this block, so not available

Can place at most 1 family (left)

Total families: 1 (row 1) + 1 (row 2) = 2

But my previous solution would have calculated:

**For row 1: left=true, middle=true, right=false → can place 2 families (incorrect, because left and middle overlap)**

For row 2: left=true, middle=false, right=false → can place 1 family

Total: 2 (row 1) + 1 (row 2) = 3 (incorrect)

**Corrected Approach**

**The issue is that the left (B-E / 2-5) and middle (D-G / 4-7) blocks overlap (seats D / 4 and E / 5 are in both). So if both are available, we can't place two families - only one can be placed in either left or middle, not both.**

**We need to modify the logic to:**

**First check if left and right are both available (no overlap, can place 2 families)**

**Otherwise, check if any single block is available (left or middle or right)**

**Corrected Solution Code**

class Solution {

public int maxNumberOfFamilies(int n, int[][] reservedSeats) {

Map<Integer, Set<Integer>> reservedMap = new HashMap<>();

// Populate the reserved seats map

for (int[] seat : reservedSeats) {

int row = seat[0];

int col = seat[1];

reservedMap.putIfAbsent(row, new HashSet<>());

reservedMap.get(row).add(col);

}

int result = 2 \* (n - reservedMap.size()); // rows with no reservations can fit 2 families

for (Set<Integer> reserved : reservedMap.values()) {

boolean left = !reserved.contains(2) && !reserved.contains(3)

&& !reserved.contains(4) && !reserved.contains(5);

boolean right = !reserved.contains(6) && !reserved.contains(7)

&& !reserved.contains(8) && !reserved.contains(9);

boolean middle = !reserved.contains(4) && !reserved.contains(5)

&& !reserved.contains(6) && !reserved.contains(7);

if (left && right) {

result += 2;

} else if (left || right || middle) {

result += 1;

}

}

return result;

}

}

**Explanation of Corrections**

**Checking left and right first**: These blocks don't overlap (B-E and F-J), so if both are available, we can place 2 families.

**Otherwise check individual blocks**: If only left, right, or middle is available, we can place 1 family.

**Middle block check**: Changed to correctly check seats 4-7 (D-G) without overlapping with left/right checks, make sure middle is only checked after left and right, since we want to achieve maximum families

This corrected solution will now return the expected output of 2 for the given input.