**Assignment I**

**PS07 Freezer Room**

**Assignment Set Number: PS07 Freezer Room**

**Group Name: Group-21**

**Contribution Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Group No** | **Name** | **Student Email id** | **% Contribution** |
| **1** | **Arun M M** | **2021SC04065** **@wilp.bits-pilani.ac.in** | **100%** |
| **2** | **DESHPANDE AADITYA PRASAD** | **2021SC04064** **@wilp.bits-pilani.ac.in** | **100%** |
| **3** | **RISAB BISWAS** | **2021SC04063** **@wilp.bits-pilani.ac.in** | **100%** |

# Introduction

## Problem Definition

An organization has a unique freezer room with easy-to-adjust temperature, lock with keys, manual defrost option, and basket or adjustable shelf etc. Employees of this organization keep moving in and out of the freezer room on work. Organization uses a unique employee ID and smart card to identify their employee’s movement to the freezer room. Whenever the employee swipes his card to the freezer room employee ID is recorded. When an employee enters the room for the first time, the counter is set to 1. From then onwards, each time an employee swipes out of the room the counter is incremented and incremented again when he enters back. If the counter is odd, it means the employee is inside the freezer room and if the counter is even, it means he/she is out of the room

The organization uses this data to perform the following analytics:

1. How many employees entered the freezer room today?
2. Number of employees that have entered the freezer room today and are currently inside?
3. Check if a specific employee is inside the freezer room or outside?
4. List of employees that have swiped (in or out) more than x number of times?
5. Which employee ids within a range of IDs entered the freezer room, the swipe counter for them, and whether they are inside or outside the freezer room?

# Choice of Data Structure

From the problem statement we get to know that the application is search intensive, that means we need to have a mechanism that will be able to retrieve data more effectively. Keeping a note that swipe entries are dynamic in nature so data structure need to accommodate these run time data, in other words size of data structure should not be limited like array.

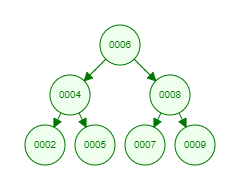
By considering all the above points use of **binary tree** will be more effective. As Binary tree has no limits on the memory, and search operation can be optimised while constructing binary tree itself. So, we will be implementing binary search tree to solve this problem.

# Implementing a Binary Tree

As program need to be search optimised, we can create a binary tree whose left child will be smaller employee ID than the root node’s employee ID and root’s right child employee ID will be greater that root node’s employee ID.

Constructing binary tree in such a way allows us to make optimum use of binary tree’s non-linear memory alignment.

For example: Employee IDs = [6, 4, 8, 4, 8, 2, 5, 7, 2, 9, 5]



When the employee ID are repeated, we increment the counter ‘attctr’ by one. This ensures us that there are no duplicate entries inserted into the tree.

# Runtime Analysis

## inFreezer Option:

For ‘inFreezer’ choice we need information from the whole tree, this is achieved by performing inorder traversal. Since the number of edges that can originate from a node is limited to 2 in the case of a Binary Tree, the maximum number of total edges in a Binary Tree is n-1, where n is the total number of nodes.

The time complexity then becomes *O(n + n-1)*, which is **O(n)**.

## checkEmp Option:

in ‘checkEmp’ option we need to search in a binary tree to get the ‘attctr’ of specified employee. Considering the worst case that all employees are arriving in a sorted order of their employee ID, that will make this tree skewed, then the search operation will be linear in nature.

The time complexity then becomes **O(n)**, where n is total number of nodes.

## freqVisit Option:

in ‘freqVisit’ option we need to search all the nodes checking for ‘attctr’ parameter having greater than or equal to the specified value. This traversal is also done inorder fashion. Since the number of edges that can originate from a node is limited to 2, the maximum number of total edges in a Binary Tree is n-1. So time complexity becomes **O(n + n-1) that is O(n)**. n is total number of nodes.

## range Option:

for ‘range’ Option we need to search for employee ID that fits in the given range. Considering the worst-case scenario, we will get a huge range that covers complete tree. So, we end up traversing the whole tree. Time complexity to traverse the whole tree in in-order fashion is *O(n + n-1)* that’s nothing but **O(n)** time.

# Alternate Way of Modelling the problem

This Problem can also be solved by using the Linked-List data structure. As Linked list data structure is also non-linear in memory alignment. This non-linear nature enables us make use of any memory location in space.

Following are the few points that will impact the performance.

## Insertion in Data Structure

For inserting in Linked list at worst case we need to insert at last element, making time complexity as *O(n)*. In binary search tree we get the insertion location in *O(log n)* time, as we split the whole tree into half per traversal. In this case Binary Tree performs better than the linked list.

## Search in Data Structure

Searching operation at worst case in both the data structures will be *O(n)* as worst case of Linked list is the last element and for binary tree if it gets skewed then its linear search. But if we consider the average time required for the search in binary tree is *Θ(log n)* as we divide the search into half, per iteration.

As discussed above this problem can also be solved using the linked list but binary tree performs better at insert and search operation.

# Results

For example: in Input file “inputPS07.txt”

6  
4  
8  
4  
8  
2  
5  
7  
2  
9  
5

inFreezer:  
checkEmp: 12  
checkEmp: 5  
freqVisit: 2  
range: 01:07

## Output in file

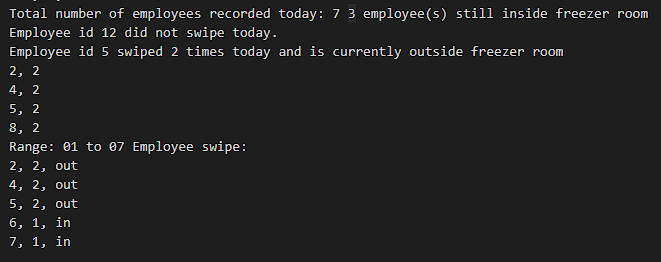


Image 1: Output in outputPS07.txt

## Output on console for inorder traversal

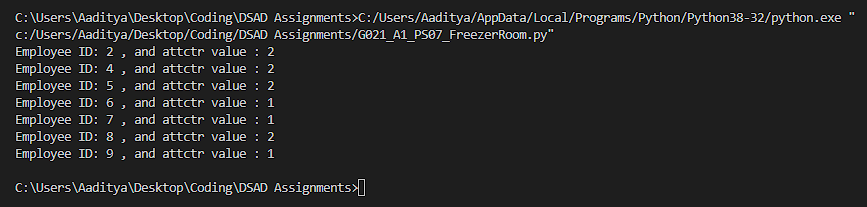


Image 2: Output on console for inorder traversal