

EMPLOYEE ATTENDANCE ANALYTICS

DSAD Assignment -PS1



January 7, 2019

**Submitted by**

Goldy Mathew Aloysious (Roll No: 2018AB04037 )  
Gowtham SM (Roll No: 2018AB04047)  
Vengal O.G.T. (Roll No: 2018AB04020)

# Problem Statement:

In an attendance monitoring system, the unique integer ID number of every employee is noted whenever the employee enters the organization. First time an employee enters into the office, the corresponding id is set to 1. From then onwards, each time an employee leaves the office premises for tea break or lunch break, and enters back this id is incremented. You can figure out that when this id is odd on a day, he is in the office premises and even when he is out.

The organization uses this data to perform the following analytics:

1. How many employees came today?
2. Did a particular employee come today?
3. How often did an employee enter into the office?
4. Which employee moves out of office most number of times?
5. Who all came within a range of IDs, and how often they entered?

# Implementation Details

We have chosen to implement AVL tree. AVL tree is a self-balancing Binary Search Tree (BST). For an AVL tree the difference between heights of left and right subtrees cannot be more than one for all nodes for an AVL tree and hence this is expected to a very efficient algorithm even if the large data set is skewed todays one side. The AVL trees are more balanced compared to Red-Black Trees. AVL trees may cause more rotations during insertion and deletion than the Red-black trees, but our application does not involve many frequent insertions and deletions, so AVL tree was the choice of implementation.

# Input data

All the integer employee ids given for input into a file named **input.txt**.

|  |
| --- |
| 23  22  41  121  41  22  41  121 |

Table 1: Input.txt data file

# Recorded Outputs

### Menu driven Operation

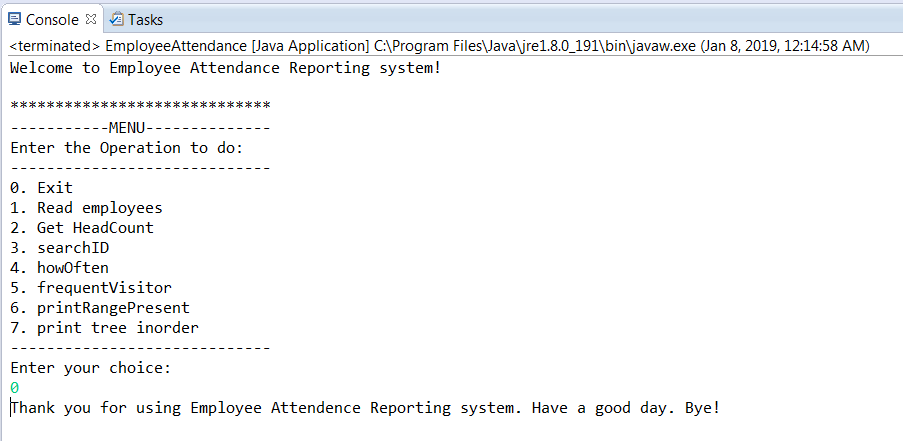
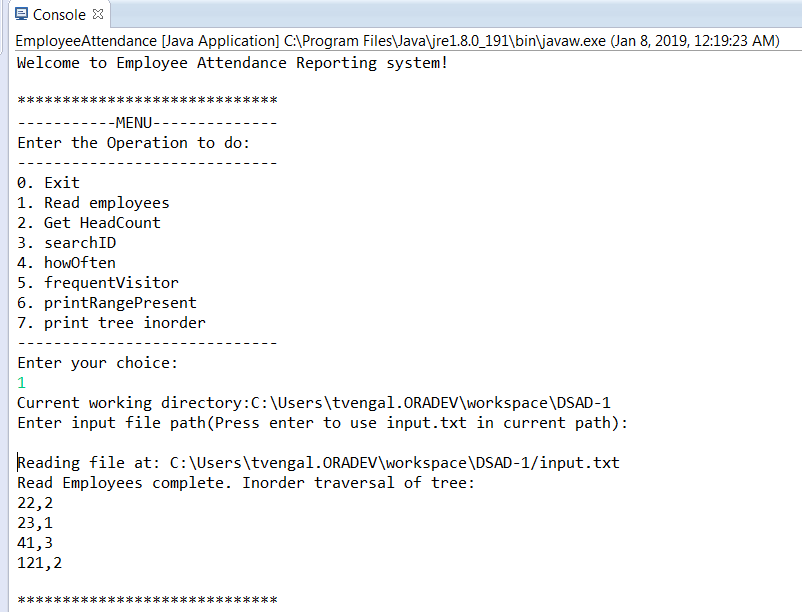


Fig 1: Screenshot of Menu choices in the application and exiting option

### Read input data



### Print Binary Tree InOrder

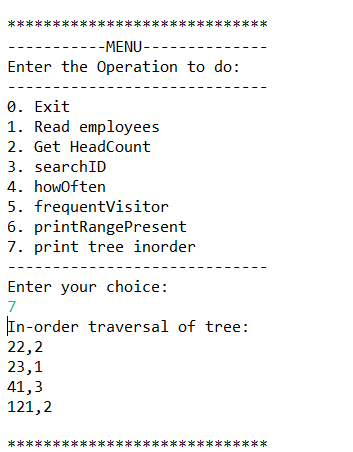


Fig 2: Screenshot of Printing Binary Tree in In-order

c. How many employees came today?

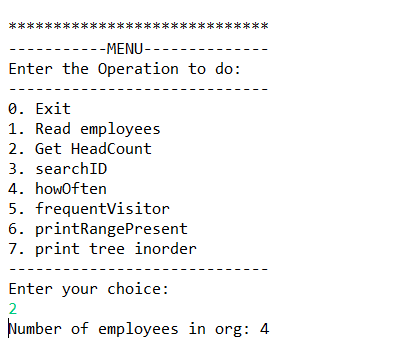


Fig 2: Printing the number of employees who came in today

d. Did a particular employee come today?

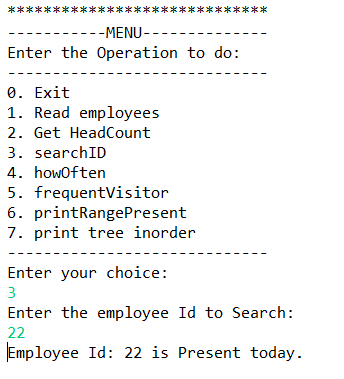


Fig 3: Finding whether a specific employee entered the office today

e. How often did an employee enter into the office?

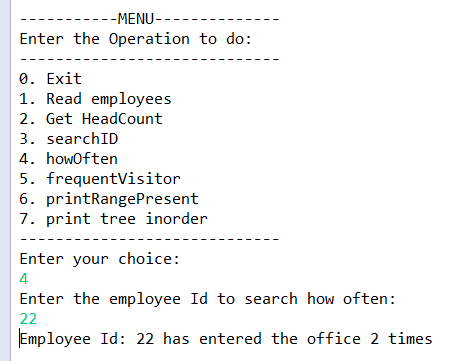


Fig 4: Finding how often a specific employee entered the office today.

f. Which employee moves out of office most number of times?

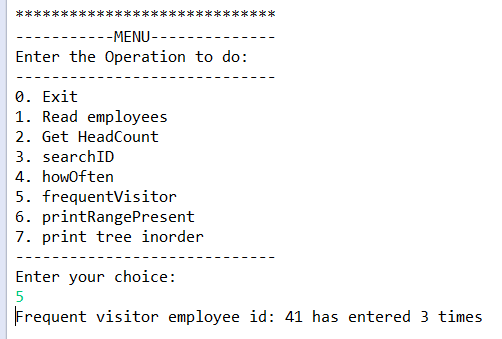


Fig 4: Finding the employee who moves frequently in and out of office.

f. Who all came within a range of IDs, and how often they entered?

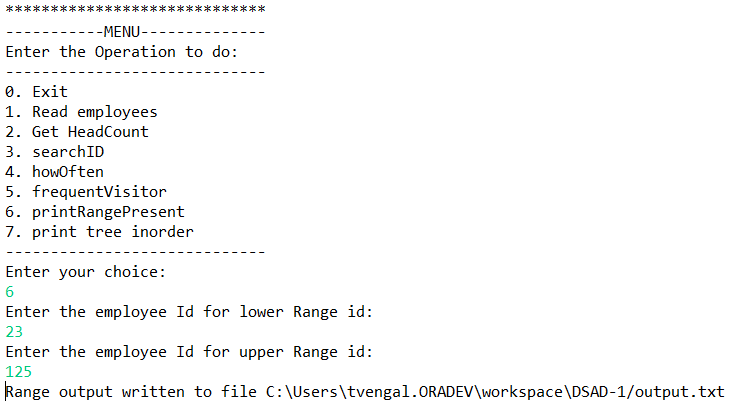


Fig 5: Output displayed in console for finding the list of employee who moves frequently in a specific employee number range

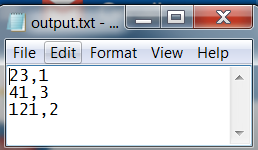


Fig 6: Output saved in a file for finding the list of employee who moves frequently in a specific range

## Conclusions

We have been able to successfully compile, execute and tested various datasets of much higher value than the sample data. We believe that this is an efficient data structure in terms of space and time for larger data sets. All the basic operations of this algorithm is done in O(log n) time.