Design and Analysis of Algorithms, CSCI 5454

Final Project, Spring 2016

Indian Railway Inquiry System based on AVL Trees

By Alimulla Shaik

University of Colorado, Boulder

Table of Contents

1. Introduction

AVL tree is a special kind of binary search tree which self balances to maintain heights of the two child subtrees of any node differ by at most one. AVL trees will be helpful and efficient in application where search operation performed more frequently than insertion. Indian railway system, is a case where new train details are not added (insertion) frequently but millions of people search for the train details daily. So instead of just implementing AVL tree algorithm, I have opted to design an application "Indian Railway Inquiry System" which efficiently uses all the operations of AVL trees.

Millions of people in india travel daily and they constantly search for the train information. This application will provide them the information they are looking for in faster and efficient manner possible.

2. Indian Railway Inquiry System using AVL Trees

I have designed an application based on AVL trees which uses operations of it including insertion, deletion and searching. Collected indian railway train details from the different sources and inserted each train details like train number, train name, source and destination station as nodes of the avl tree. Train information is inserted in the avl tree based on train number. After every insertion and deletion operation, need to rebalance the nodes of the tree to ensure properties of the avl tree.

2.1 Operations

Indian Railway Inquiry System application performs 3 major operations listed below.

Insertion: Application can insert any number of train details. As mentioned earlier each train detail is inserted as node in the avl tree and rebalances nodes to ensure avl tree height property (heights of the two child subtrees of any node differ by at most one). This operation will be useful to update the system with new train details in the future. User need to enter details like train number, train name, source and destination station to update particular train details into the system. If any of that information is missing then system will report error message to the user that particular field is mandatory, otherwise update the system with entered train details.

Deletion: Application can delete any number of train details. When it is required to delete any particular train information from the system, this operation will be useful and after that rebalances nodes to ensure avl tree height property. When user entered train number to delete for the information in the system, application will perform search operation in the avl tree for the node equal to the train number entered. If train information is found then the corresponding node will be deleted from the avl tree and system, otherwise returns error message that train information could not found in the system to delete.

Search: Application can search for train information from the system. Search operation will be easy as all the nodes in the avl tree are balanced. When user entered train number to search for the information in the system, application will perform search operation in the avl tree for the node equal to the train number entered. If train information is found then the same will be displayed to the user, otherwise returns error message that train information could not found in the system.

Insertion operation with Example:

We will create a new node for a new train entry which is supposed to be updated in the system and will insert in appropriate position. After the insertion, go back upto the root node and update heights of the nodes. If any node violates avl height property then need to rebalance the node by rotation method. Single or double rotations may be required to balance the tree.

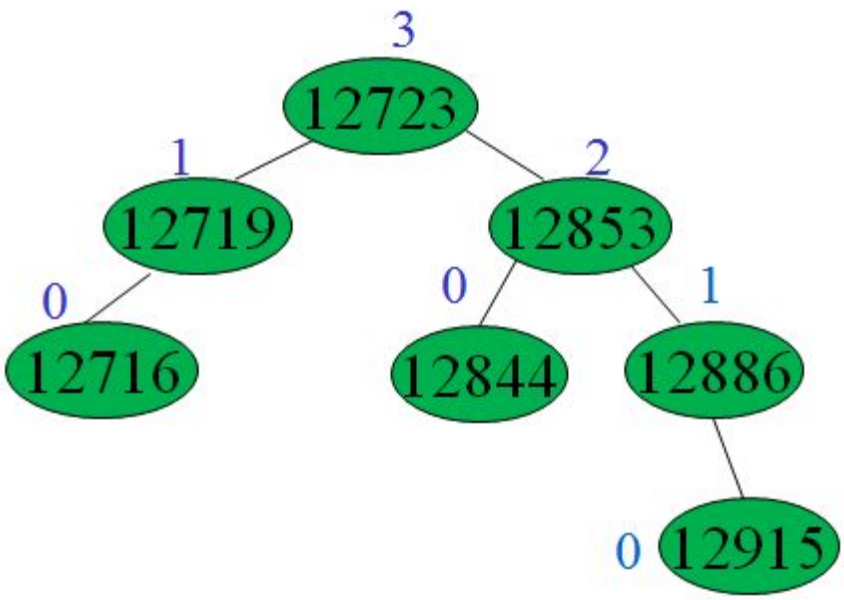
Four cases are possible during insertion and are listed below.
Let the node that needs rebalancing be 'x'.
Cases which require single rotation:

1. Insertion into left subtree of left child of 'x'.
2. Insertion into right subtree of right child of 'x' .

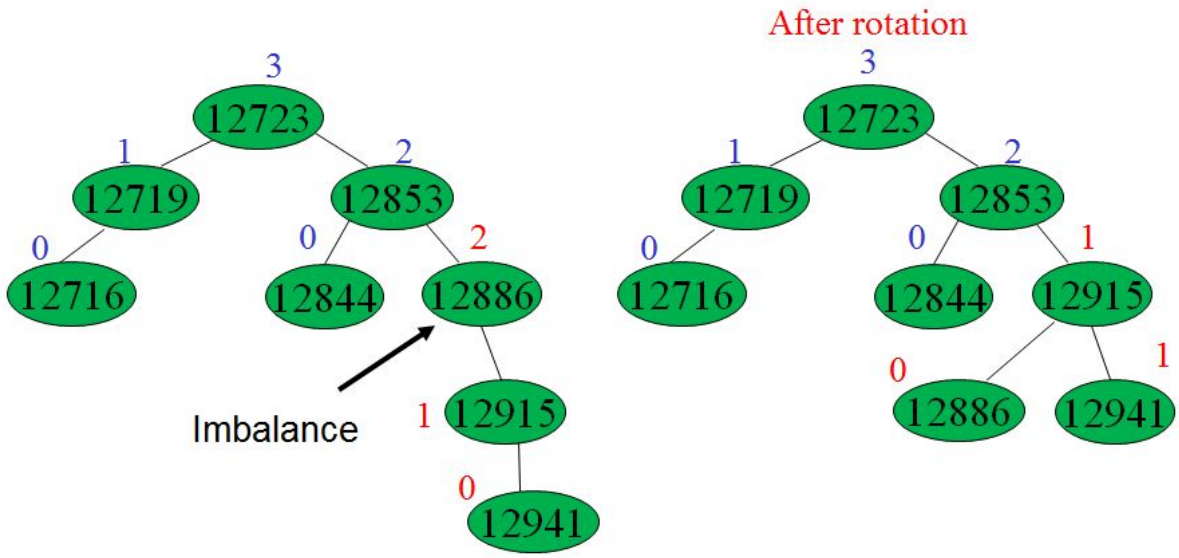
Cases which require double rotation:

3. Insertion into left subtree of right child of 'x'.
4. Insertion into right subtree of left child of 'x' .

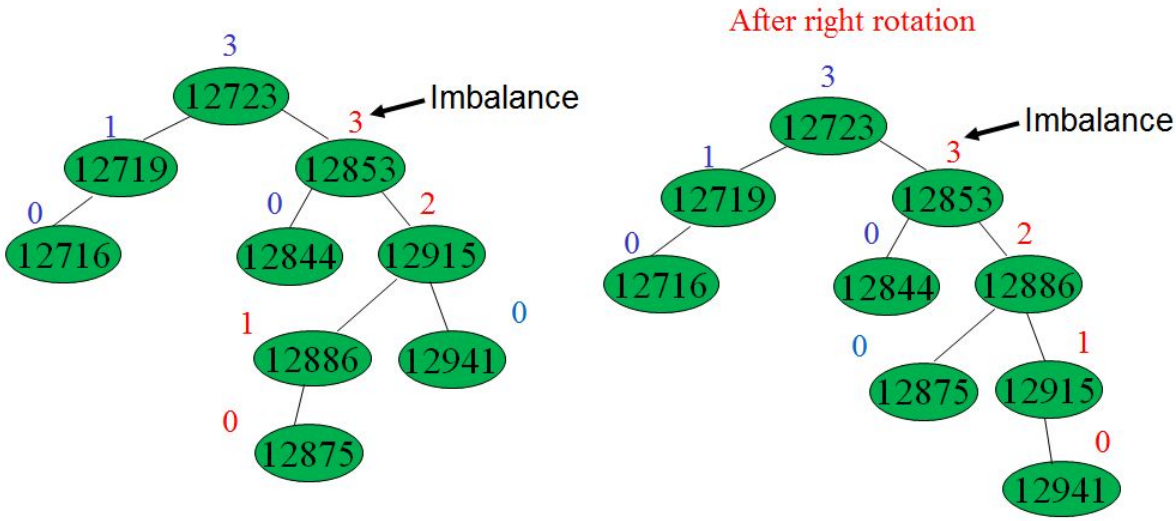
Consider the below example from our system, each train information is stored as node and showed height of the node as well (in blue color next to node).

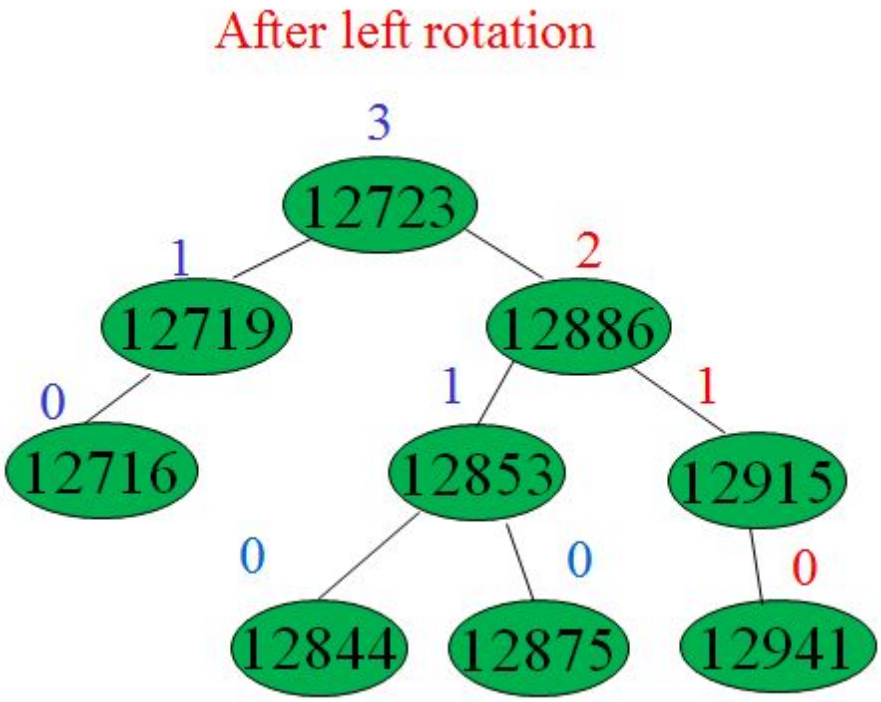


Now insert 12941 train information into the system. After insertion system will look like below. As you can see there is an imbalance at node 12855. We need to perform left rotation (single rotation) at the node 12886 to rebalances this tree.



Now insert 12875 train information to the above tree. After insertion, you can observe tree imbalance as shown below. For this case we need to make double rotation to rebalance the tree. First right (at node 12915) and then left rotation (at node 12853) is performed as shown in below to rebalance the tree.





Deletion:

Once user entered train information which need to eliminated from the system, application will search for the node in the avl tree and will delete the corresponding node. One deletion of the node can imbalance several of its ancestors. So when we delete any node which causes imbalance on the node's parent, we need to make necessary rotation at the parent node and need to traverse all the way up the ancestry line, checking for balance and perhaps make some more rotations if required to fix the avl tree. System will make rotations similar to the insertion cases described above.

3. Correctness

If the user wants to know about the source and destination stations of a particular train, then our application will return the same details if entered train number is valid and present in the railway system. Otherwise it throws an error message that train details does not exist in the system. The same is followed for the insertion and deletion as explained above. This application works for all the cases including boundary cases where double rotation is required to rebalance the tree or system.

This application handles the basic case where inserting new train information at the left or right node of the subtree and in this case avl tree is balanced even after new node is inserted so no need to rebalance any node. In the worst case we need to rebalance most of the nodes in the tree after insertion or deletion and our application takes care of this case as well. If the entered train details are not valid or not present in the system then application throws error message and informs about the same to the user. Required fields like train number need to entered while searching or deleting particular train information. If required

fields are missed and did not enter by the user then application throws an error message to communicate the same to the user. Our application takes care of most of the boundary cases and all the possible error scenarios and even validated all the fields. So we can generalize this application that it works well for any given situation.

4. Run Time Analysis

Our application works based on operations of the avl tree. AVL tree with 'n' nodes has O(logn) height.

Insertion:
AVL tree insertions are same as binary search insertions with at most two possible rotations. Rotations only reassign left, right and parent of the node so it takes O(1) time. Binary search tree insertions take time equivalent to the height of the tree and avl tree has height of O(logn). So insertion operation of AVL tree takes O(logn)+O(1) = O(logn) time.

Search:
During search, number of nodes remaining to consider are halved with each node considered. So search operation takes O(logn) time.

Deletion:
For deletion, we may need to make O(logn) more rotations after deleting the node but since rotations are O(1) operations so additional rotations does not affect overall run time of O(logn) time.

AVL Tree Operations	**Best**	**Average**	**Worst**
Insertion	O(logn)	O(logn)	O(logn)
Search	O(logn)	O(logn)	O(logn)
Deletion	O(logn)	O(logn)	O(logn)

As our application based on the avl tree operations so it also have the same complexity or running time as avl tree for all the operations. Our application need to store all the nodes of the tree and for n nodes O(n) space required to perform necessary operations. Running time and space complexity is mentioned below for our application.

Our Application	**Best**	**Average**	**Worst**
Insertion	O(logn)	O(logn)	O(logn)
Search	O(logn)	O(logn)	O(logn)

Deletion	O(logn)	O(logn)	O(logn)
Space Complexity	O(n)	O(n)	O(n)

5. Numerical Characterization of the performance

Explored and collected indian railway train details from the different sources to develop this application. Each node in the application has different details about the particular train. These values are varied from single digit to multiple digits and built the application keeping in mind that it can hold any type of value. As of now this, I have stored values up to 10^5 in this application but it can hold any values without any restriction. Earlier versions of Python used to have limit for int variables. But, this is dropped and now Python treats integer as an object. So, python allocates 32 bit for the value object reference is pointing to, but as the value goes beyond 2^{32} it can keep moving up all the way up to the size of RAM on your computer. So this application based on python, can hold any values up to the size of the RAM of computer.

Insertion: Inserting into left or right of the subtree (check the insertion example figure above) is easy and does not need any rebalancing of the node after insertion operation. It yields best performance in this application. For example, insert the input with train number "332" belongs to this case and gives best performance. The worst case is when we try to insert the input whose value is close to the root node may imbalance all the nodes all the way upto root node and need to rebalance almost entire half of the tree. For example, insert the input with train number "12721" (closes to root node "12723") belongs to this case and gives the worst performance. Space required in both the cases is same as storing all the nodes possible in the tree. For 'n' nodes we need O(n) space required to store all the nodes.

Search: Searching the node which is near to root node is easy and return results in best time possible. For example, search the input with train number "12724" belongs to this case and gives best performance. The worst case is If the node is present in one of the leaf nodes then may need to traverse entire half of the tree to get the node. For example, search the input with train number "334" belongs to this case and gives worst performance. Space required in both the cases is same as storing all the nodes possible in the tree. For 'n' nodes we need O(n) space required to store all the nodes.

Deletion: Deleting the left or right of the subtree is easy and does not need any rebalancing of the node after deletion operation. It yields best performance in this application. For example, Delete the input with train number "334" belongs to this case and gives best performance. The worst case is when we try to delete the input whose value is close to the root node may imbalance all the nodes all the way down to leaf nodes and need

to rebalance almost entire half of the tree. For example, delete the input with train number "12719" (closes to root node "12723") belongs to this case and gives the worst performance.
Space required in both the cases is same as storing all the nodes possible in the tree. For 'n' nodes we need O(n) space required to store all the nodes.

6. Enhancements

This application considered only indian railway details and can be extend it to support different countries train information. Now only indian people can utilize the advantages of this application and can get the train information. Different countries have different types of trains and we can collect all the necessary information to help the people of particular country in searching train information. In that case application need to handle huge sets of information. As we are using AVL trees to store the information, application still provides efficient results in faster manner possible.

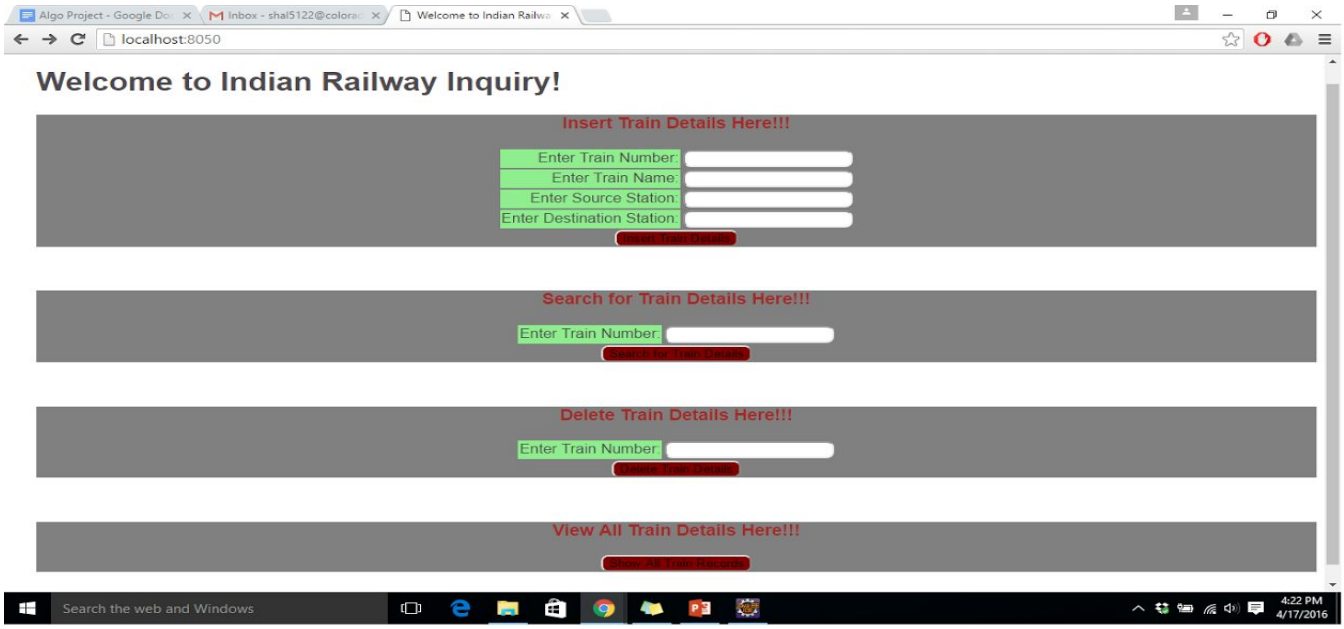
7. References

1. https://en.wikipedia.org/wiki/AVL_tree
2. Bottle framework used for UI. http://bottlepy.org/docs/dev/index.html

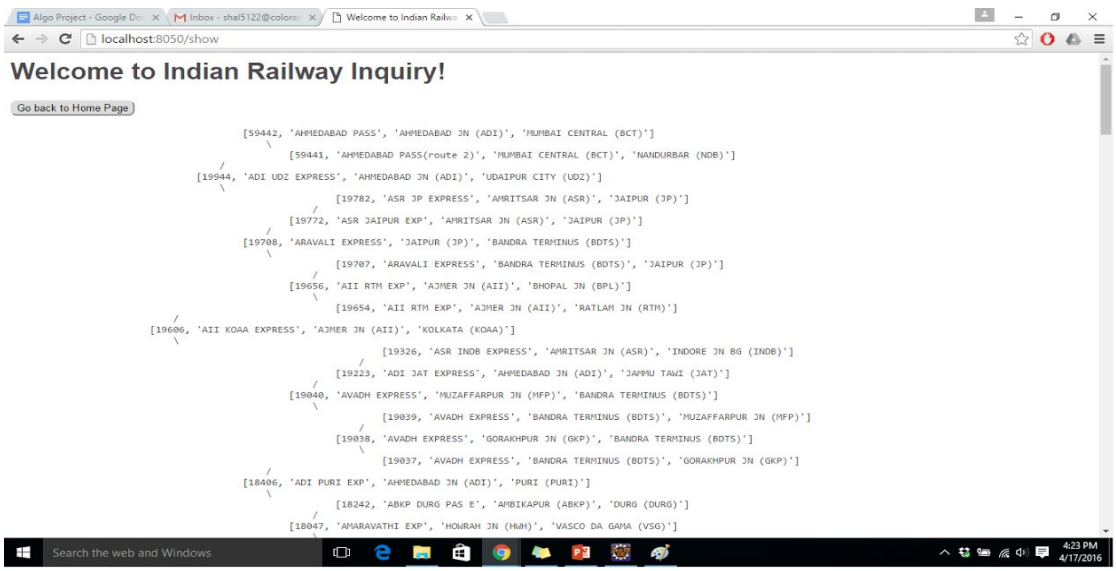
8. Results

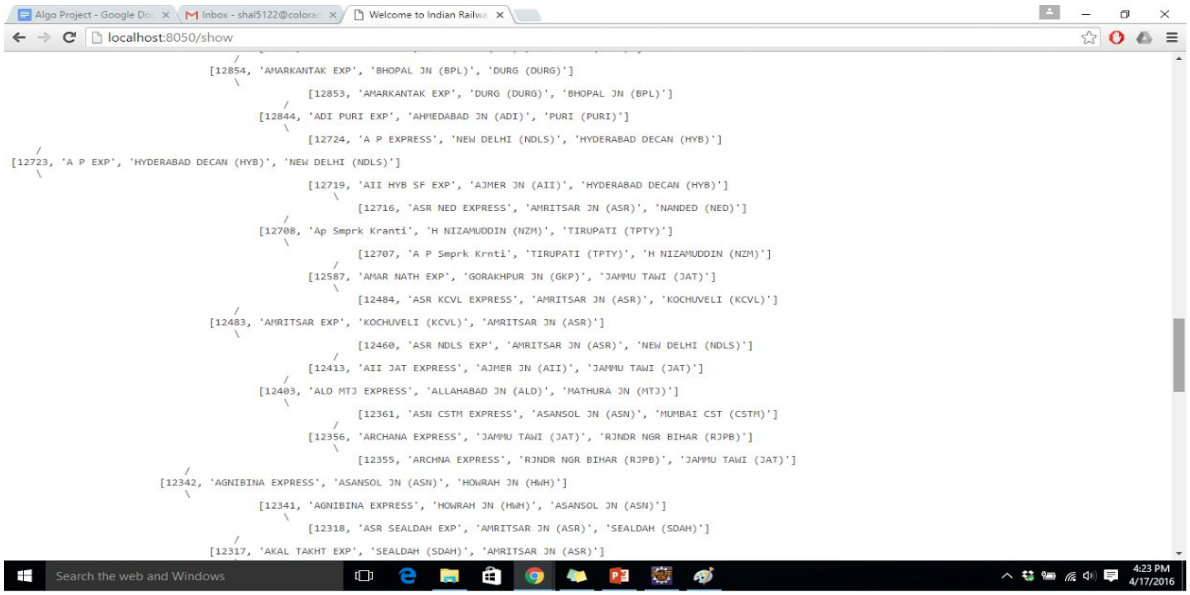
Screenshots for the different results of the application are posted below.

a. Home page of the application is shown below.

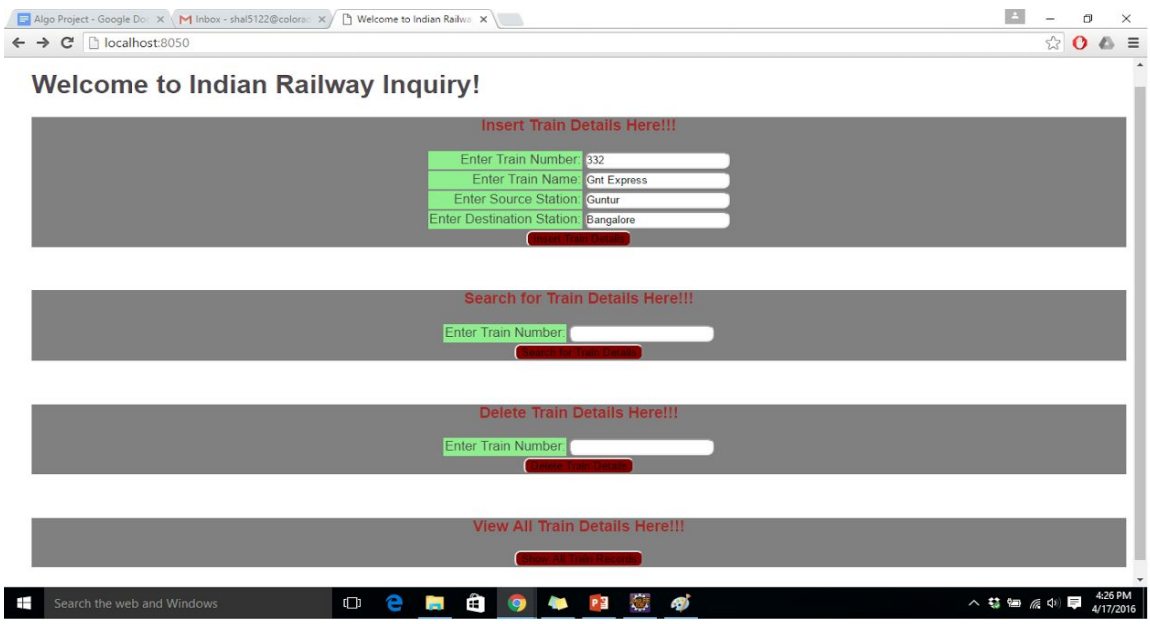


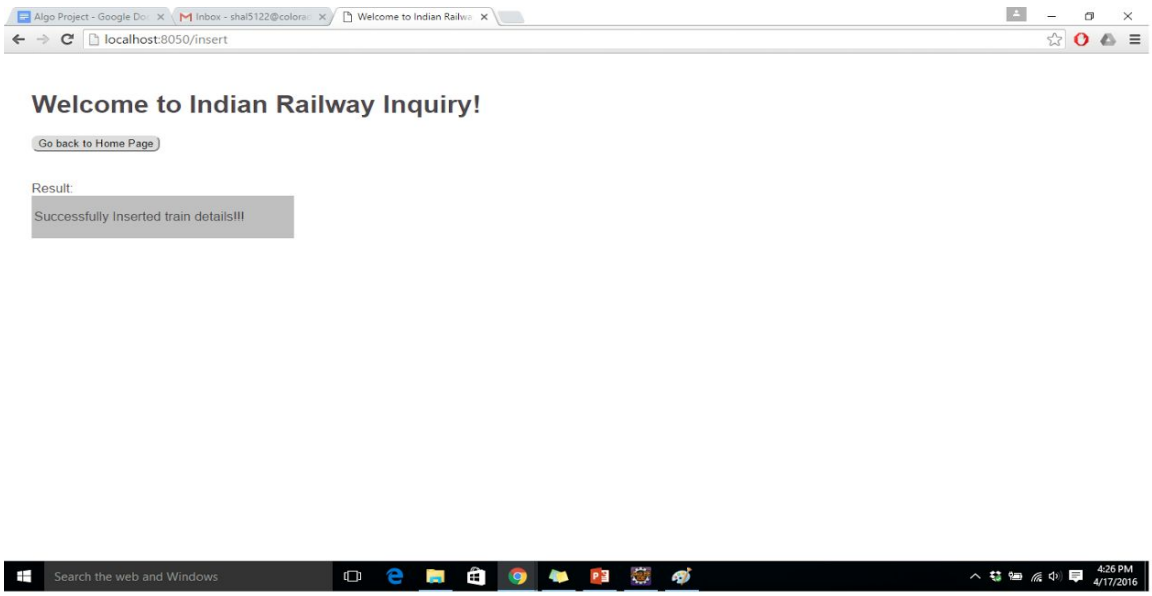
b. We can view all the train details present in the system using “Show All Train Records” option in the home page. Once that option is selected, system will redirect to below page which displays all the train information. Same details are shown below.



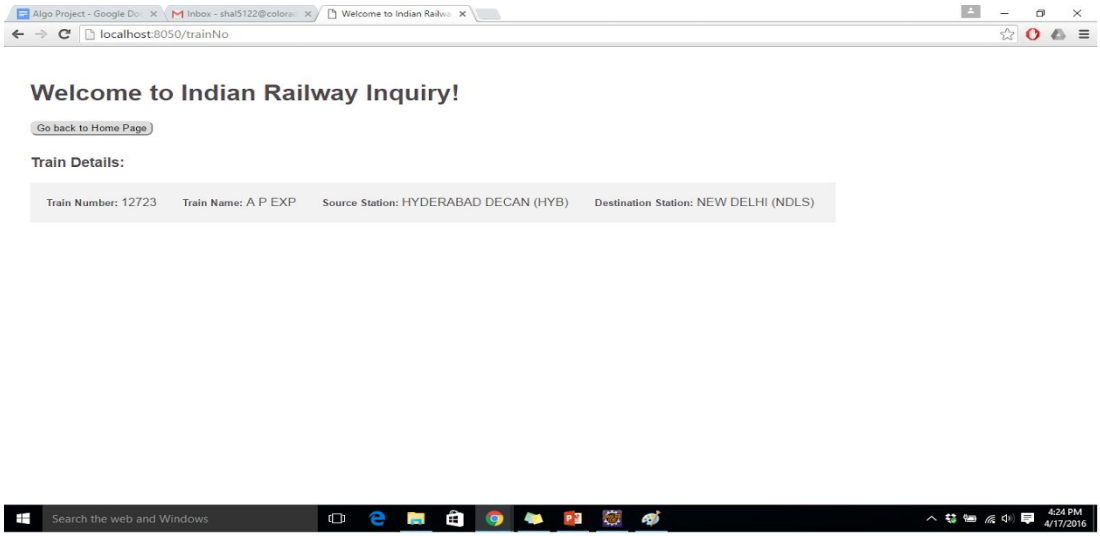


c. User can insert any train details using option “Insert Train Details” in the home page but before that user need to provide train related information like train number, name etc. Same is shown below.

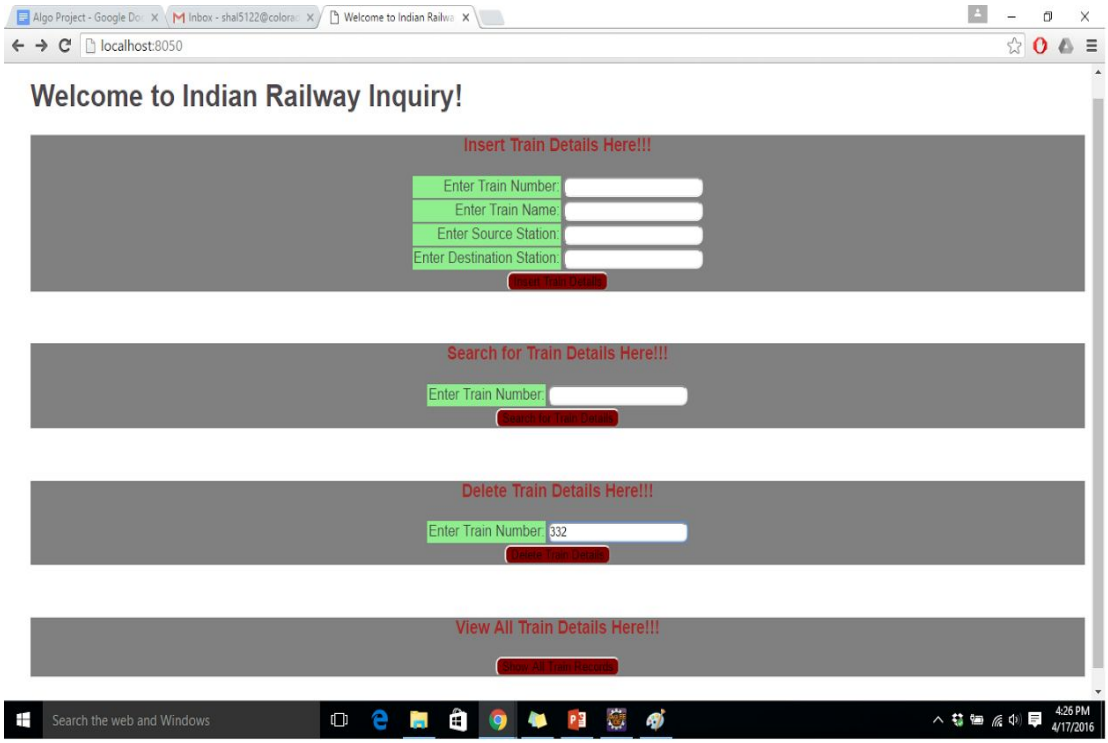


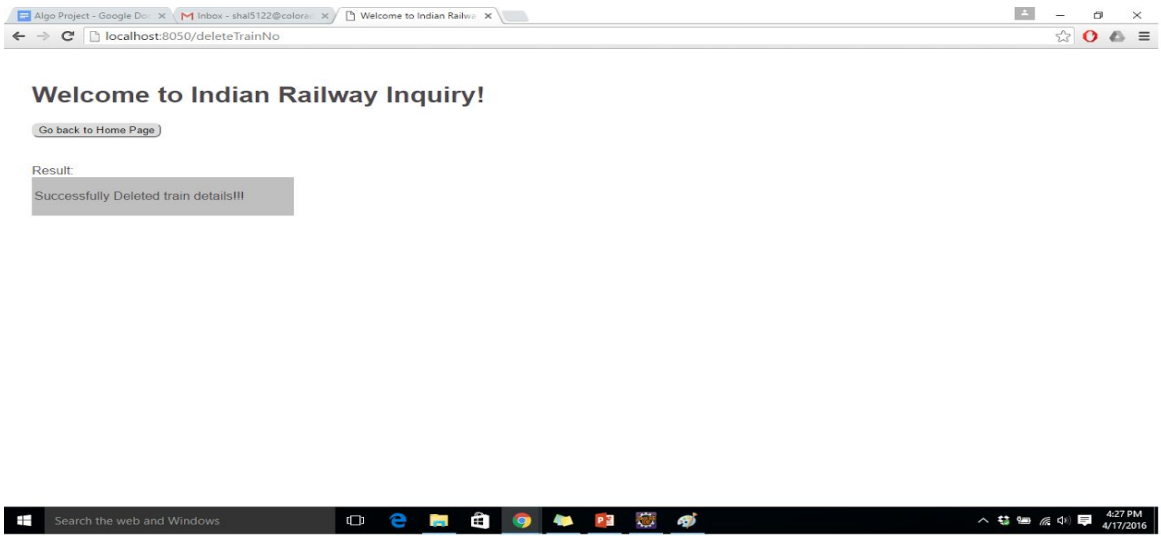


d. User can search for particular train information using option “Search for Train Details” in the home page (shown below). But user need to provide required train number to search in the system.



e. User can delete any particular train information using option “Delete Train Details” in the home page (shown below). But user need to provide required train number to delete it from the system.





f. All fields in the home page are mandatory and required to perform corresponding operation. When user entered missed any fields then system will notify an error message to communicate the same to the user and asks the user to fill out the particular field.


Algo Project - Google Do
Inbox - shal5122@colora
Welcome to Indian Railwa
localhost:8050
Welcome to Indian Railway Inquiry!
Insert Train Details Here!!!
Enter Train Number:
Enter Train Name:
Enter Source Station:
Enter Destination Station:
Please fill out this field.
Insert Train Details
Search for Train Details Here!!!
Enter Train Number:
Search for Train Details
Delete Train Details Here!!!
Enter Train Number:
Delete Train Details
View All Train Details Here!!!
Show All Train Records
Search the web and Windows
4:24 PM
4/17/2016