Project 2 - Internet

CmpE 250, Data Structures and Algorithms, Fall 2022

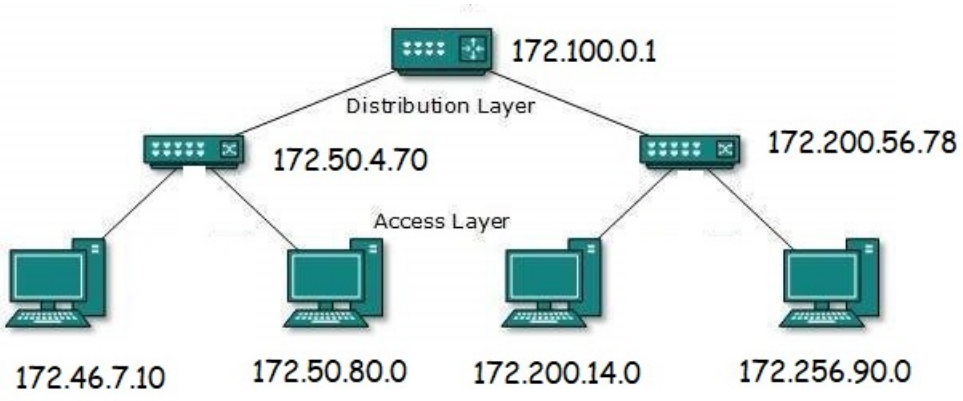
Instructor: Özlem Durmaz İncel
TAs: Suzan Ece Ada, Barış Yamansavaşçılar
SAs: Batuhan Çelik, Bahadır Gezer, Zeynep Buse Aydın, Ömer Faruk Ünal

Due: 4/11/2022, 23:59 Sharp

1 Introduction

Have you ever wondered how the Internet works? The implementation of the internet is quite similar to a postal service. When you try to send a package from BOUN to METU, firstly you go to the post office in Hisarüstü, then, they send their package to gathering center in Levent and then to the gathering center of Istanbul. From there, the package is sent to the gathering center in Ankara, Çankaya, METU, and then to the receiver. The Internet works with the exact same principals, where specialized computers named routers act like the postal service. When you send a Whatsapp message, the message first goes to your modem, which is a router, then to the router responsible for your neighbourhood, district, city, country in order. There is a different structure in international space but notice that, up to the country node, with different cities, districts, and neighbourhoods the topology is a tree topology!

One question comes to mind is that, how does the addressing work in the internet. You probably heard about the IP addressing. IP address uniquely identifies each device(Node) connected to a network. Overall, network looks like this with addresses:



2 Objective

Your objective in this project is modelling this networking structure using different data structures: BST and AVL tree. We will be giving you details about the nodes, then your model should:

- Add and delete nodes.

- Send a message between nodes, tracing the path the message took.
- Re-balance the tree as the number of nodes increase.
- Keep the logs of the operation.

You can safely assume that input files will be set up so that each IP will be unique.
In the end, you will be given a huge input file to observe the speed difference between BST and AVL.

2.1 Model details

Each node is uniquely identified by an IP address. For illustration valid IP addresses are used but assume any string can be used as an address(you can name your nodes like: Özlem, Ece, Barış, etc). These addresses shall be compared using `compareTo(String s)` function of a string object.

As the operations happen and messages pass between nodes, each node prints a log message containing details about the operation to a output file. You can find the log string definitions bellow.

3 Functionality and Input File

Each input file has a single IP address in the starting line, this IP is the IP of the initial root. The lines bellow that contain instructions your model should handle.

3.1 Add a new Node

Instruction: `ADDNODE <IP_ADDRESS>`

During addition of a new node, its place is found using binary search. All the nodes on the path from root node to parent node of the newly added node shall add the event to their log messages, and others should not change their log files. Log message structure for adding a new node is:
`<Logging_node_IP>: New Node being added: <IP_Address>`

3.2 Delete a node

Instruction: `DELETE <IP_ADDRESS>`

When a node is deleted, there are 2 cases: The node is a leaf node and deleted immediately or the node is not a leaf node and upon deletion, a leaf node takes its place. For this project, you are expected to use the smallest node from right subtree to replace the deleted node.

In terms of logging, logging of the delete operation is performed by the parent node of the deleted node. For the 2 cases, there are 2 possible log lines:

- When a leaf node is deleted, its parent logs:
  `<Logging_node_IP>: Leaf Node Deleted: <Deleted Node IP>`
- When a non-leaf node is deleted:
  `<Logging_node_IP>: Non-Leaf Node deleted; removed: <Deleted Node IP> replaced: <Replacing Node IP>`

Notice that only the parent of actually deleted node's parent performs a logging operation, so when you swap the intermediate node with a leaf node to remove the leaf node, the parent of the leaf node should not log anything.

3.3 Send Message

Instruction: `SEND <SENDER_IP_ADDRESS> <RECEIVER_IP_ADDRESS>`

Tree is a connected graph, meaning there is a path between each node in the topology, so it is possible to send a message between each node. However, a node can transmit a message only to its child nodes or the parent node. Thus, to arrive to the receiver, the message must jump between nodes. These message hops are logged in the nodes as:

`<Logging_node_IP>: Transmission from: <NODE_IP_THE_MESSAGE_COMES_FROM> receiver: <RECEIVER_IP> sender: <SENDER_IP>`

For example, consider the tree structure from the introduction. A message sent from 172.46.7.10 to 172.200.56.78 follows the following path:

`172.46.7.10` → `172.50.4.70` → `172.100.0.1` → `172.200.56.78`

This transmission is logged in 3 different ways in different type of nodes:

- Sender(172.46.7.10):
  `<Logging_node_IP>: Sending message to: 172.200.56.78`
- Receiver(172.200.56.78):
  `<Logging_node_IP>: Received message from: 172.46.7.10`
- Intermediate node(172.100.0.1):
  `<Logging_node_IP>: Transmission from:172.50.4.70  receiver: 172.200.56.78 sender: 172.46.7.10`

One another thing to keep in mind is that, we want you to commute the message in the shortest path, so a message should not go over the same node 2 times. (Pro tip: throw an exception when this happens for easy debugging)

3.4 Re-Balancing

Balance functionality will not be provided as instructions in the input file. Instead, your AVL tree needs to rebalance itself as the new nodes are added and some nodes are deleted. Your AVL tree should never be in a unbalanced situation. As you know, there are 2 types of rebalancing in AVL trees, single rotation and double rotation. These are just reorganization of pointers in short. When the rotations happen, all nodes involved in the rotation, 2 nodes in single rotation case and 3 nodes in double rotation case, will log the rotation type as: `Rebalancing <Rotation_type>`
Note: Check for balance deficiencies in a bottom-up manner.

On the case of BST tree, this functionality won't be used so nothing will be logged since there is no rebalancing.

4 Input File

Your program shall take the input file as the first system argument. The input file structure and an example input file is in the following:

```
<Parent_IP_Address>
Instruction 1
Instruction 2
...
```

For more, refer to the input files provided with the description.

5 Output File

Your second system argument will specify an output file, assume it is output.txt for illustration. Your file shall create 2 files, output_AVL.txt and output_BST.txt and each tree shall log to their respective output file.

You will be provided with example input files and their expected output files.

6 Submission

You will be submitting a zip file containing your code via Moodle. We will be running your code via followind commands:

```
javac Main.java
java Main <input_file> <output_file>
```

You will be delivered a huge file containing millions of instructions. In the last part of submission, you will switch on and off AVL property, then observe the time difference between AVL tree and BST tree. You can measure the time your code took via running:

```
time java Main <input> <output>
```

Note, you must run the following command in Windows to get the time command working:

```
function time { $Command = "$args"; Measure-Command { Invoke-Expression $Command 2>&1 out-default} }|
```

In the last part of your submission, add a file named `time_difference` to your zip. This file contains 2 lines: first being the time taken with AVL tree and the second being time taken with BST tree. You shall take the real time stated by your OS in Unix systems and miliseconds in Windows systems.

7 Warnings

- All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code. Any sign of cheating will be penalized by at least -100 points at first attempt and disciplinary action in case of recurrence.
- Make sure you document your code with necessary inline comments and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long. This is very important for partial grading.
- Make sure that the white-spaces in your output is correct. You can disregard the ones at the end of the line.
- Please use the discussion forum at moodle for your questions, and check if it is already answered before asking.