CS29003 ALGORITHMS LABORATORY
ASSIGNMENT 6
Date: 29th Aug, 2019

Important Instructions

1. **List of Input Files:** Routes.txt, Query.txt
2. **Output Files to be submitted:** ROLLNO_A6_output.txt, ROLLNO_A6_P1.c/.cpp, ROLLNO_A6_P2.c/.cpp
3. **THERE WILL BE PART MARKING.**
4. **Program 1 and Program 2 should have separate main functions.**
5. Files must be read using file handling APIs of C/C++. Reading through input redirection isn't allowed.
6. You are to stick to the file input output formats strictly as per the instructions above. Failing to do so might cause you to lose marks.
7. Submission through **.zip files aren't allowed**.

Futurama Interplanetary Express

Intro

You have just joined a new startup, Futurama Interplanetary Express. The company delivers cargo from one planet to another. It's currently serving 26 planets and has a station on each planet. The company's CEO, Vegeta, has just called you in his room for an urgent meeting. When you enter the meeting room you see that the CTO, Saitama, is also there. Saitama has generated a list of allowed routes that the company ships are allowed to take. The list was generated in a hurry, so it may contain some errors. You are given the task of writing a program that will filter the routes and store them efficiently. Along with this, you are also given the task of writing an AI program that will provide alternate routes in case the route does not exist.
Before any of the spaceships goes on a delivery, you would need to check if it's in the allowed route. It is clear that you needed a search index to accomplish this and Binary Search Trees (BSTs) handle this sort of task very well.
An AVL Tree is a type of "self-balancing" BST, which attempts to ensure that the ideal and average performance of a BST is guaranteed by ensuring ideal conditions are maintained. When implemented as an AVL Tree, a BST can be guaranteed to be highly time efficient, which is necessary for this task.
* For this assignment you can assume that the name of each planet is an alphabet in the English vocabulary, A-Z (Uppercase only).

Objectives of this assignment

In this assignment, you have to prepare a height-balanced tree (AVL) to store the routes from one planet to another (provided as a string of uppercase alphabets). The routes provided are in random order and could contain duplicates. This assignment will have two parts. In part I, you will have to create an AVL tree to store the filtered routes. In the second part, you will have to create a search function which will check if a given route exists in the tree or not. In case it does not exist, you will have to provide alternate routes. Both the parts are explained below.

Part I - Generating the AVL tree

As a first step you would need to store the routes that Saitama provided (Routes.txt) in an efficient data structure. Lucky for you, you remember about BST and AVL trees from your Algorithm's Lab in B.Tech and realize that it could be used for this purpose.

Step I : Error in the routes

As the route list was generated in a hurry, there were some errors. It seems that some of the planets were repeated consecutively when they were entered. For example, consider the route "EEGIDDDADB", it contains two E's and three D's consecutively. This is not allowed on a route. In such case, you will have to remove all the same consecutive alphabets and replace them with a single occurrence. So the above example after the filtering will become "EGIDADB".

Step II : Create the AVL Tree

After you have removed the errors in the route, your next task will be to add the route to the AVL tree.
The format of the routes file is as follows:

FILE: *Routes.txt*

```
10
RITK
AYADD
TALINP
AY
TIR
MORLA
TALINP
EEGIDAB
EEGKID
MORLA
```

The first line contains the number of routes available. From this file you will need to read each line (which represents the routes) and add them to the AVL Tree. The set of symbols are limited to uppercase alphabets.

Data structure

The basic data structure should follow the template below.

```
typedef struct _treenode {
      char *route;
      struct _treenode *left;
      struct _treenode *right;
      int balancefactor;
   } AVLtreenode;

typedef AVLtreenode *AVLtree;
```

NOTE:

- ▷ Before you enter any route in the tree, you will have to ensure that it's not already present. In case it's a duplicate entry, you should ignore it.

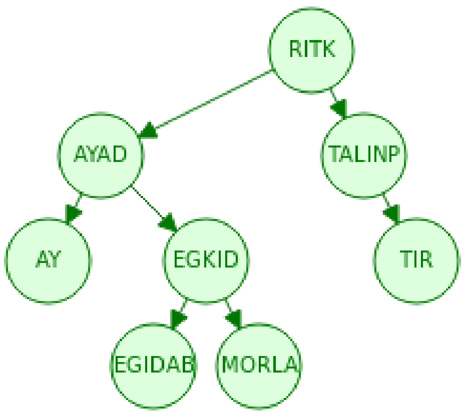

Figure 1: AVL tree generated for the input routes provided

- ▷ Each route will be a node in the AVL Tree.
- ▷ You are NOT allowed to use array to store or manipulate the AVL Trees. Failing to do so will cause you to lose marks.

Output of Part I:
For the output of the first part, you will have to print the height of the AVL tree in the first line and then the in-order traversal of the tree in the following lines.

HAL 9001 for route suggestion

As a next step for your company, you need to create HAL 9001, an AI that can find if a given route is invalid and suggest alternatives. In this case, you decided to start with a simple rule based AI.
You are provided with a file with a list of routes that the spaceships will take (Query.txt). You will provide this as an input to your AI, HAL 9001. HAL will check for the route in your AVL tree and return “Found” or “Not Found”, based on whether the route is present or absent. In case the route is absent, HAL will have to provide some alternative based on the following rules:

Rule 1: Addition
You can add one more planet in the given route to check if it’s present in the tree. You can add this to any position in the route.
Rule 2: Deletion
You can remove any of the planet from the route and check if it’s present. When you remove a planet, all the occurrence from the given route are removed.

NOTE:
The Query.txt file can contain the same type of errors as Routes.txt. So you would have to first apply the same error checking routine as you did in the previous part, before checking if a route is present.

Output of Part II:
As output of Part II, you should append the following lines to your output.txt file (created in Part I).

- ▷ You would have to first apply the error checking routine and report the corrected route.

```
Corrections for the route "EEGKIDP" : EGKIDP
```

▷ You would then have to report if the route was found in the tree or not in the following form:

```
Searching for the route "EGKIDP" : not found
```

▷ In case the route is not found, you would have to apply each of the rule defined above and report the possible routes. In case of multiple routes they should be separated by space. If no route is found using a rule, leave it blank.

```
Alternate route suggestion:
Rule 1:
Rule 2: EGKID
```

Example

A complete set of sample input and output files is made available through moodle, please check attachments.

FILE: *Routes.txt*

```
10
RITK
AYADD
TALINP
AY
TIR
MORLA
TALINP
EEGIDAB
EEGKID
MORLA
```

FILE: *Query.txt*

```
3
TALNP
MORLA
EEGKIDP
```

FILE: *Output.txt*

```
4
AY
AYAD
EGIDAB
EGKID
MORLA
RITK
TALINP
TIR
Corrections for the route "TALNP" : TALNP
Searching for the route "TALNP" : Not Found
Alternate route suggestion:
Rule 1:TALINP
Rule 2:

Corrections for the route "MORLA" : MORLA
Searching for the route "MORLA" : Found

Corrections for the route "EEGKIDP" : EGKIDP
Searching for the route "EGKIDP" : Not Found
Alternate route suggestion:
Rule 1:
Rule 2:EGKID
```