CS29206 Systems Programming Laboratory, Spring 2022–2023

Programming Assignment 4

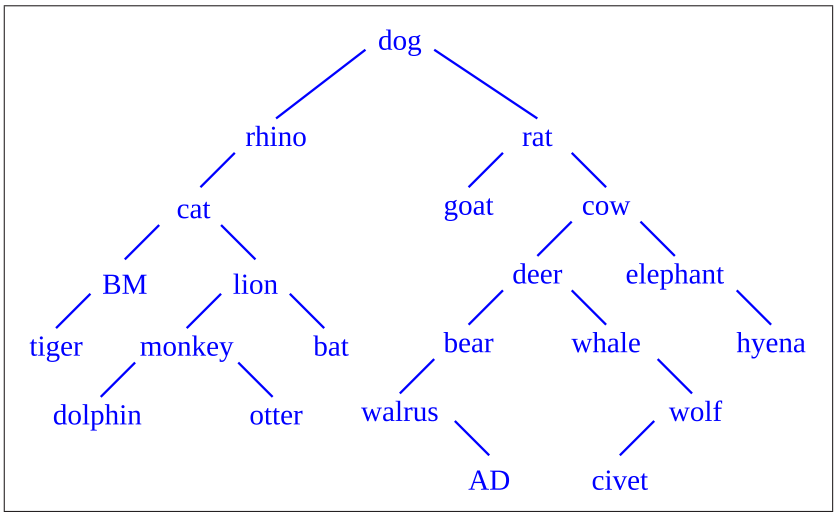
Date: 23–March–2023 (to be submitted in the lab)

Topic: Programming in gawk

Consider a binary tree T with each node storing a string as the key. Assume that each key is of length $\leqslant 10$. Being only a binary tree (not a search tree), there need not be any ordering of the keys stored in the nodes in the tree. Although this is not the general situation, assume for this assignment that the keys stored in the nodes are distinct from one another. If that is the case, we can specify the tree in a text file in the following format. The first line of the file stores the number of nodes in T (and nothing else). This is followed by a list of the nodes and their children, one line for each node. If a node stores *key*, and its two child nodes (left and right) store the keys *lkey* and *rkey*, the file specifies that node as:

key:*lkey*,*rkey*

If (exactly) one of the two child nodes does not exist, an empty string is used as *lkey* or *rkey* (as the case is). A leaf node has no child, and is not listed in the file. Assume that no key or no line contains any space (or tab). Only : and , are used as separators. The following picture shows a binary tree on the names of some mammals (the keys) and the representation of the tree in a text file **`mtree.txt`** (to be supplied to you).



```
22
bear:walrus,
BM:tiger,
cat:BM,lion
cow:deer,elephant
deer:bear,whale
dog:rhino,rat
elephant:,hyena
lion:monkey,bat
monkey:dolphin,otter
rat:goat,cow
rhino:cat,
walrus:,AD
whale:,wolf
wolf:civet,
```

The (non-leaf) nodes can appear in any order in the text file (here the lexicographic order is used which has nothing to do with the structure of the tree, but even that need not be the case). Moreover, the ordering of the keys in neither the file nor the tree bears any BS or CS significance (BS = Biological Science).

In this assignment, you need to write an **executable** gawk script that reads the text file (the file name is to be supplied in the command line), creates the tree using the data structures supplied by gawk, and does some simple operations on the tree.

Since gawk does not support pointers as in C, you maintain the following arrays. Let n denote the number of nodes in the tree (available as the first line of the input text file).

`H[]` In this assignment, the nodes store mutually distinct keys, but this is not the general case. With this in mind, assign the integer-valued IDs $1, 2, 3, \ldots, n$ to the n nodes. The **associative array** **`H[]`** stores the mapping of the keys (strings) to these integer-valued IDs. As you read the input file, keep on assigning the IDs in the sequence $1, 2, 3, \ldots$ to the *new* keys found. In our example, you have **`H["bear"] = 1`**, **`H["walrus"] = 2`**, **`H["BM"] = 3`**, **`H["tiger"] = 4`**, and **`H["cat"] = 5`**. Then, you encounter the key **`BM`** which is already assigned an ID, so skip this key. Continue the numbering as **`H["lion"] = 6`**, **`H["cow"] = 7`**, **`H["deer"] = 8`**, **`H["elephant"] = 9`**, **`H["whale"] = 10`**, and so on.

`K[]` This is a usual array indexed by the integer-valued IDs. The entry **`K[i]`** is the key (a string) which is given the ID **`i`**. You must have **`K[H[key]] = key`**, and **`H[K[i]] = i`** for all valid keys **`key`** and for all valid IDs **`i`**. In our example, **`K[5] = "cat"`**, **`K[2] = "walrus"`**, **`K[8] = "deer"`**, and so on.

`L[]` This is an integer-indexed and integer-valued array. For **`i`** in the range $1,2,3,\ldots,n$, **`L[i]`** should store the **ID** of the left child of the node with ID **`i`**.

`R[]` This is again an integer-indexed and integer-valued array. For **`i`** in the range $1,2,3,\ldots,n$, **`R[i]`** should store the **ID** of the right child of the node with ID **`i`**.

`P[]` This is yet another integer-indexed and integer-valued array. For **`i`** in the range $1,2,3,\ldots,n$, **`P[i]`** should store the **ID** of the parent of the node with ID **`i`**.

If a child or the parent of a node **`i`** does not exist, the corresponding entry **`L[i]`**, **`R[i]`**, or **`P[i]`** may be left undefined or set to an integer (like 0) outside the range $1,2,3,\ldots,n$. The choice is yours.

All these five arrays should be fully populated in the input-file-reading section (write only one such section).

Write three functions that process the binary tree T built in the reading section.

- Write a function **`prntree()`** that prints the information of all the n nodes in the tree. You should follow the printing format as illustrated in the Sample Output. For the alignment, assume that $n < 1000$, and each key is a string of length at most 10. Use **`printf()`**. The keys may appear in any order, but all the keys must be printed including those stored in the leaf nodes (they do not appear in the input file).
- Write a function **`findroot()`** to find the **ID** of the root node in the tree. Use the array **`P[]`** for this purpose, and note that the root is the only node having no parent.
- Write a function **`inorder()`** to print the inorder listing of the keys in the tree. Follow the format given in the Sample Output. Print only five keys in each line (except perhaps the last). The outermost call of this recursive function is supplied the ID of the root node, obtained by **`findroot()`**. The navigation in the tree should use the arrays **`L[]`** and **`R[]`**. The array **`K[]`** is also needed for printing the keys (as strings) instead of the integer-valued IDs.

The END section should call the above three functions and then stop.

Submit a **single file** (the executable gawk script that you have written). The script should have a BEGIN section (defining FS and doing other things, if any, that you need), a *single* input-file-reading section, the utility functions as described earlier, and an END section that calls the utility functions.

Sample Output

```
Going to read tree with 22 nodes

        BM :  ID =   3  Left child = tiger       Right child = -
       bat :  ID =  16  Left child = -           Right child = -
     rhino :  ID =  12  Left child = cat         Right child = -
        AD :  ID =  20  Left child = -           Right child = -
       rat :  ID =  13  Left child = goat        Right child = cow
     tiger :  ID =   4  Left child = -           Right child = -
      deer :  ID =   8  Left child = bear        Right child = whale
  elephant :  ID =   9  Left child = -           Right child = hyena
       ...

The root node has ID 11 and key "dog"

Inorder listing of the keys:
     tiger          BM         cat    dolphin     monkey
     otter        lion         bat      rhino        dog
       ...

Bye...
```
