Arbol Programming Language Proposal

Andrea Mary McCormick, Anthony Palmeira Nascimento,
Derek Hui Zhang, Andreas Cheng
{amm2497, asp2199, dhz2104, hc3142}@columbia.edu

January 31, 2021

1 Introduction

In the realm of programming there is a plethora of languages that enable the implementation of trees. However, in many of those cases such implementations are not native to the language and that poses a challenge to inexperienced programmers. The purpose of the ARBOL language is to abstract much of the logic behind this data structure in order to reduce the complexity of working with trees. Our language will include the most common tree operations such as traversals, enumerating tree, enumerating subtree, searching for a node, adding a new node at a specific position, deleting a node, pruning (removing subtree), grafting (adding subtree), finding root for any node, finding the lower common ancestor of two nodes. With this new layer of abstraction offered by ARBOL, students will be able to solve complex algorithmic problems without having to worry about properly implementing or operating on the tree data structure.

2 Overview of Arbol

Arbol is a Python-like, dynamically typed, and imperative language that provides a more intuitive binary tree manipulation. As the foundation of many powerful data structures, such as Binary Search Tree and Huffman Coding Tree, the Binary tree helps us locate or compress data more effectively. Arbol's syntax is based on Python, but it comes with more syntactic sugar to make building binary trees easier.

Our goals are:

- Create a language with a short learning curve.
- Help programmers to use binary tree easier syntactically.

3 Language Details

3.1 Keywords

Arbol’s reserved keywords:

`def, True, False, Null, while, for, continue, break,`
`case, pass, if, elif, else, in, class, print,`
`int, float, char, bool, preorder, inorder, postorder`

3.2 Data Types and Operations

Arbol’s primitive types are integers, floats, booleans, and strings. Programmers can represent a character with a single character string.

Data Type	Description	Operators	Examples
int	4 bytes	=, ==, !=, +=, -=, +, -, *, **, /, ˆ, %, >, <, >=, =<	2030-9=2021 # False 2021!=2020 # True 4+=1 # 5

float	8 bytes		2020.5+.5 # 2021.0
			12.5>12.05 # True
			1==1.00001 # False

boolean	True/False	=, ==, !=, !, &&, ||	1&&0 # False 1 || 0 # True

String	Immutable	=, ==, !=, +=,
		+, *, >, <, >=, =<
			a = ”left”
			b = ”right”
			a+b = ”leftright”
			a*2 = ”leftleft”

3.3 Control Flow

Arbol’s control flow works like Python.

3.3.1 while loop

```
while True:
    print("Infinity loop")
```

3.3.2 for loop

```
for e in [7,2,1][::-1]:
    puts(e)
#Output: 127
```

3.4 Functions

Arbol’s function works like Python.

```python
def pow(a, b):
    return a**b
```

```python
def sumByCondition(f, nums):
    ans=0
    for n in nums:
        if f(n):
            ans+=n
    return ans
```

3.5 Comments

Arbol's comments works like Python. However, unlike PEP8, Arbol favors both single-line and mutiple-line comments.

3.5.1 Single line comment

```python
# I'm a comment.
```

3.5.2 Mutiple line comment

```python
"""
We
Are
All
Comments.
"""
```

3.6 Memory

Arbol pass arguments by name.

4 Optional Types

4.1 Standard Library

Our standard library will include the following functions:

- height(node), isleaf(node), isRoot(node)
- for node in node.children(tree):

  See the Object Oriented Programming section below for more details.
- dfs(node, goal), bfs(node, goal)

Beside general trees, we will also implement Python-like list. We will add more standard library functions as we see fit.

4.2 Object-Oriented Programming

Arbol supports simple object with class.

```
class Apple:
    def eat:
        this.hp -= 5
    def grow:
        this.hp +=2
```

5 Examples

```
import treeLib.ab

# Python-like print (Python3)
print("Hello world!")

# Creating a root for a binary tree
bTree = barbol("A")

# Make Node left and Node right as the leaves of root
Node("B") <$ bTree $> Node("C")

# Preorder / inorder / postorder Traversal
for n preorder bTree:
    print(n)

for n inorder bTree:
    print(n)

for n postorder bTree:
    print(n)


# defining function
def gcd(int a, int b):
    while b:
        a, b = b, a % b
    return a
# References:
# https://en.wikipedia.org/wiki/Euclidean_algorithm
# https://www.geeksforgeeks.org/gcd-in-python/
```