# Simple Binary Search Tree Documentation

***Release 0.4.1***

**Adrian Cruz**

October 23, 2014

Contents

Contents:

CHAPTER 1

# Simple Binary Search Tree

Simple Binary Search Tree is a simple implementation of a binary search tree

- Free software: MIT license
- Documentation: https://simplebst.readthedocs.org.

## 1.1 Features

To use Simple Binary Search Tree in a project:

```
# At minimal, you'll need to import simplebst.Node
from simplebst import Node

# Create a single element tree with value of 23
# Its left and right sub-trees are set to None
tree = Node(23)
```

Get the value of a Node:

```
tree.get_value()
```

Get the left/right child Node of a Node:

```
tree.get_left()
tree.get_right()
```

Insert a new node into the tree:

```
# Import simplebst.utils.insert_node
from simplebst.utils import insert_node

# Insert a node will modify the tree you specify
# So, we'll use our previous example of "tree"
insert_node(tree, 17)

# If you were curious you should see the correct
# value if you do the following
tree.get_left().get_value()

# Let's fill the tree with values
for value in [18, 27, 53, 11]:
    insert_node(tree, value)
```

For more detailed usage, please see the usage documentation: https://simplebst.readthedocs.org/en/latest/usage.html

CHAPTER 2

# Installation

At the command line:

```
$ easy_install simplebst
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv simplebst
$ pip install simplebst
```

CHAPTER 3

# Usage

To use Simple Binary Search Tree in a project:

```
# At minimal, you'll need to import simplebst.Node
from simplebst import Node

# Create a single element tree with value of 23
# Its left and right sub-trees are set to None
tree = Node(23)
```

Get the value of a Node:

```
tree.get_value()
```

Get the left/right child Node of a Node:

```
tree.get_left()
tree.get_right()
```

Insert a new node into the tree:

```
# Import simplebst.utils.insert_node
from simplebst.utils import insert_node

# Insert a node will modify the tree you specify
# So, we'll use our previous example of "tree"
insert_node(tree, 17)

# If you were curious you should see the correct
# value if you do the following
tree.get_left().get_value()

# Let's fill the tree with values
for value in [18, 27, 53, 11]:
    insert_node(tree, value)
```

In-order traversals

in_order_nodes generator:

```
from simplebst.traversals import in_order_nodes

# Use a generator to get all nodes in-order
for node in in_order_nodes(tree):
    print(node.get_value())
```

```
# You _should_ get the following:
# 11
# 17
# 18
# 23
# 27
# 53
```

in_order_list:

```
from simplebst.traversals import in_order_list

# We need to store the values in a list,
# so we'll create an empty one
ordered_list = []

# in_order_list will modify ordered_list with
# Node()'s from the tree in-order
in_order_list(tree, ordered_list)

# You can now iterate the ordered_list
for node in ordered_list:
    print(node.get_value())
```

Level-order traversals

level_order_nodes generator:

```
from simplebst.traversals import level_order_nodes

# Use a generator to get all nodes in level-order
for node in level_order_nodes(tree):
    print(node.get_value())

# You _should_ get the following:
# 23
# 17
# 27
# 11
# 18
# 53
```

level_order_list:

```
from simplebst.traversals import level_order_list

# level_order_list() returns a list
# so we can iterate it like so
for node in level_order_list(tree):
    print(node.get_value())
```

Pre-order traversals

pre_order_nodes generator:

```
from simplebst.traversals import pre_order_nodes

# Use a generator to get all nodes in pre-order
for node in pre_order_nodes(tree):
    print(node.get_value())
```

```
# You _should_ get the following:
# 23
# 17
# 11
# 18
# 27
# 53
```

pre_order_list:

```
from simplebst.traversals import pre_order_list

# We need to store the values in a list,
# so we'll create an empty one
pre_ordered_list = []

# pre_order_list will modify pre_ordered_list with
# Node()'s from the tree in pre-order
pre_order_list(tree, pre_ordered_list)

# You can now iterate the ordered_list
for node in pre_ordered_list:
    print(node.get_value())
```

Post-order traversals

post_order_nodes generator:

```
from simplebst.traversals import post_order_nodes

# Use a generator to get all nodes in post-order
for node in post_order_nodes(tree):
    print(node.get_value())

# You _should_ get the following:
# 11
# 18
# 17
# 53
# 27
# 23
```

post_order_list:

```
from simplebst.traversals import post_order_list

# We need to store the values in a list,
# so we'll create an empty one
post_ordered_list = []

# post_order_list will modify post_ordered_list with
# Node()'s from the tree in post-order
post_order_list(tree, post_ordered_list)

# You can now iterate the ordered_list
for node in post_ordered_list:
    print(node.get_value())
```

Useful utility functions

tree_height:

```
from simplebst.utils import tree_height

# Get the height of the tree we've been using
height = tree_height(tree)
print(height)

# You should get 2
```

CHAPTER 4

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 4.1 Types of Contributions

### 4.1.1 Report Bugs

Report bugs at https://github.com/drincruz/simplebst/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

### 4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

### 4.1.4 Write Documentation

Simple Binary Search Tree could always use more documentation, whether as part of the official Simple Binary Search Tree docs, in docstrings, or even on the web in blog posts, articles, and such.

### 4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/drincruz/simplebst/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 4.2 Get Started!

Ready to contribute? Here's how to set up *simplebst* for local development.

1. Fork the *simplebst* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/simplebst.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv simplebst
   $ cd simplebst/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 simplebst tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

## 4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/drincruz/simplebst/pull_requests and make sure that the tests pass for all supported Python versions.

## 4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_simplebst
```

CHAPTER 5

# Credits

## 5.1 Development Lead

- Adrian Cruz <”drincruz at gmail dot com”>

## 5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

# History

CHAPTER 7

## 0.1.0 (2014-09-11)

- First release on Github.

CHAPTER 8

## 0.2.0 (2014-09-19)

- Code cleanup and updated utils and traversals

CHAPTER 9

## 0.3.0 (2014-10-08)

- **Added the following traversals:**
  - Pre-order
  - Post-order
  - Level-order

CHAPTER 10

## 0.4.0 (2014-10-16)

- Added tree_height() util
- Added insert_node() unit tests that I missed previously (yay for code coverage!)

## 0.4.1 (2014-10-22)

- Updated setup.py to support nose as its test suite So `'python setup.py test'` can be run successfully.

CHAPTER 12

# Indices and tables

- *genindex*
- *modindex*
- *search*