pyalgos Documentation

Release 0.1.2

Bharadwaj Yarlagadda

September 13, 2016

Contents

pyalgos is a package for implementing various algorithms:

- Sorting
  - Insertion Sort
  - Selection Sort

CHAPTER 1

Links

- Project: https://github.com/bharadwajyarlagadda/pyalgos
- Documentation: http://pyalgos.readthedocs.io
- Pypi: https://pypi.python.org/pypi/pyalgos
- TravisCI: https://travis-ci.org/bharadwajyarlagadda/pyalgos

CHAPTER 2

Features

- Supported on Python 2.7 and Python 3.3+.

CHAPTER 3

Guide

3.1 Installation

You can install pyalgos with all the latest changes:

```
$ git clone git@github.com:bharadwajyarlagadda/pyalgos.git
$ cd pyalgos
$ python setup.py install
```

3.2 User's Guide

3.2.1 Sorting

Insertion Sort

A simple sorting algorithm that builds the final sorted list of items - one item at a time. It can be less efficient on larger lists.

- Best Case: Input is already sorted out. This takes only `O(n)` running time. For ex, [1, 2, 3, 4, 5] - in this case it takes only linear time to sort out the items.
- Average/Worst Cases: In both the cases, it takes `O(pow(n, 2))` running time to sort out the items. In both these cases, the items in the list are either in a reverse order (or) zig-zag order.

You can use `Insertion Sort` from `pyalgos` for both list of integers and list of strings.

```
from pyalgos.sorting import insertion

numbers_sorted = insertion([3, 4, 2, 7, 5, 1])
alphabets_sorted = insertion(['a', 'z', 'y', 'b', 'w'])

assert numbers_sorted == [1, 2, 3, 4, 5, 7]
assert alphabets_sorted == ['a', 'b', 'w', 'y', 'z']
```

Note: You can provide either list/tuple of values.

Selection Sort

A sorting algorithm that build the final sorted list of items. It is inefficient on larger lists.

- Best/Average/Worst Cases: In all these cases, the running time is `O(pow(n, 2))`.

You can use `Selection Sort` from `pyalgos` for both list of integers and list of strings.

```
from pyalgos.sorting import selection

numbers_sorted = selection([3, 4, 2, 7, 5, 1])
alphabets_sorted = selection(['a', 'z', 'y', 'b', 'w'])

assert numbers_sorted == [1, 2, 3, 4, 5, 7]
assert alphabets_sorted == ['a', 'b', 'w', 'y', 'z']
```

Note: You can provide either list/tuple of values.

3.3 API Reference

3.3.1 Sorting

`pyalgos.sorting.`**`insertion`**(*elements*)

Returns the sorted values using insertion sort algorithm.

The best case is when the input is an array that is already sorted. In this case, insertion sort has a linear running time O(n).

The worst case is when the input is an array sorted in reverse order. In this case, insertion sort has a quadratic running time O(pow(n,2)).

Parameters **`elements`** (*`list/tuple`*) – A list/tuple of values provided by the user.

Returns A list/tuple of elements sorted in ascending order.

Return type list/tuple

New in version 0.1.0.

Changed in version 0.1.1: Added validation for checking whether every element in the list is a string (when strings are provided in the list).

Changed in version 0.1.2: Added support for tuples. Now the user can also provide a tuple of values.

`pyalgos.sorting.`**`selection`**(*elements*)

Returns the sorted values using selection sort algorithm.

Selection sort has O(pow(n, 2)) time complexity in all the three cases (Best, Average and Worst).

Parameters **`elements`** (*`list/tuple`*) – A list/tuple of values provided by the user.

Returns A list/tuple of elements sorted in ascending order.

Return type list/tuple

New in version 0.1.0.

Changed in version 0.1.1: Added validation for checking whether every element in the list is a string (when strings are provided in the list).

Changed in version 0.1.2: Added support for tuples. Now the user can also provide a tuple of values.

CHAPTER 4

Project Info

4.1 License

4.2 Versioning

This project follows Semantic Versioning.

It is recommended to only use or import objects from the main module, pyalgos.

4.3 Changelog

4.3.1 v0.1.2

- Added support for `tuple` of values in both `insertion()` and `selection()` sorting methods.

4.3.2 v0.1.1

- Added validation for checking whether all the elements in a given list are strings or not in both `insertion()` and `selection()` sorting methods.

4.3.3 v0.1.0

- First release.
- Added `insertion()` method for implementing insertion sort algorithm.
- Added `selection()` method for implementing selection sort algorithm.

4.4 Authors

4.4.1 Lead

- Bharadwaj Yarlagadda, yarlagaddabharadwaj@gmail.com, bharadwajyarlagadda@github

4.5 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:

4.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/bharadwajyarlagadda/pyalgos.

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.

Fix Bugs

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

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" or "help wanted" is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bharadwajyarlagadda/pyalgos.

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 :)

4.5.2 Get Started!

Ready to contribute? Here's how to set up `pyalgos` for local development.

1. Fork the `pyalgos` repo on GitHub.
2. Pull your fork locally:

```
$ git clone git@github.com:<username>/pyalgos.git
```

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

```
$ cd pyalgos
$ pip install -r requirements-dev.txt
```

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 linting and all unit tests by testing with tox across all supported Python versions:

```
$ invoke tox
```

6. Add yourself to `AUTHORS.rst`.
7. Commit your changes and push your branch to GitHub:

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

8. Submit a pull request through the GitHub website.

4.5.3 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 README.rst.
3. The pull request should work for Python 2.7, 3.4, and 3.5. Check https://travis-ci.org/bharadwajyarlagadda/pyalgos/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

I

S