pyalgs Documentation

Release 0.0.14

Xianshun Chen

May 26, 2017

Contents

pyalgs is a library of python implementation for algorithms in the "Algorithms" book (http://algs4.cs.princeton.edu/home/) and Robert Sedgwick's Algorithms course using Python (Part I and Part II)

The main purpose of this library is to provide a companion library for python developers who are learning the algorithms in the "Algorithms" book

CHAPTER 1

Installation:

To install the package using pip:

```
$ pip install pyalgs
```

CHAPTER 2

Usage

To use the algorithms or data structures in your python code:

```
from pyalgs.data_structures.commons.stack import Stack

stack = Stack.create()
stack.push(10)
stack.push(1)

print [i for i in stack.iterate()]

print stack.is_empty()
print stack.size()

popped_item = stack.pop()
print popped_item
```

CHAPTER 3

Features

- Algorithms and data structures introduced in the “Algorithms” book.
- Test coverage for each algorithm and data structure.

CHAPTER 4

Tests:

the unit tests of all algorithms and data structures can be run with the following command from the root folder:

```
$ python -m unittest discover -s . -p "*_unit_test.py"
```

CHAPTER 5

Contributing:

Contributions are always welcome. Check out the contributing guidelines to get started.

CHAPTER 6

Table of Contents:

Data Structures

Stack

```
from pyalgs.data_structures.commons.stack import Stack

stack = Stack.create()
stack.push(10)
stack.push(1)

print [i for i in stack.iterate()]

print stack.is_empty()
print stack.size()

popped_item = stack.pop()
print popped_item
```

Queue

```
from pyalgs.data_structures.commons.queue import Queue

queue = Queue.create()
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)

print [i for i in queue.iterate()]

print queue.size()
print queue.is_empty()
```

```
deleted_item = queue.dequeue())
print deleted_item
```

Bag

```
from pyalgs.data_structures.commons.bag import Bag

bag = Bag.create()

bag.add(10)
bag.add(20)
bag.add(30)

print [i for i in bag.iterate()]

print bag.size()
print bag.is_empty()
```

Priority Queue

Minimum Priority Queue

```
from pyalgs.data_structures.commons.priority_queue import MinPQ

pq = MinPQ.create()
pq.enqueue(10)
pq.enqueue(5)
pq.enqueue(12)
pq.enqueue(14)
pq.enqueue(2)

print pq.is_empty()
print pq.size()

print [i for i in pq.iterate()]

deleted = pq.del_min()
print(deleted)
```

Maximum Priority Queue

```
from pyalgs.data_structures.commons.priority_queue import MaxPQ

pq = MaxPQ.create()
pq.enqueue(10)
pq.enqueue(5)
pq.enqueue(12)
pq.enqueue(14)
pq.enqueue(2)

print pq.is_empty()
```

```
print pq.size()

print [i for i in pq.iterate()]

deleted = pq.del_max()
print deleted
```

Symbol Table

Symbol Table using Binary Search Tree

```
from pyalgs.data_structures.commons.binary_search_tree import BinarySearchTree
bst = BinarySearchTree.create()

bst.put("one", 1)
bst.put("two", 2)
bst.put("three", 3)
bst.put("six", 6)
bst.put("ten", 10)

for key in bst.keys():
    print(key)

print bst.get("one"))
print bst.contains_key("two")

print bst.size()
print bst.is_empty()

bst.delete("one")
```

Symbol Table using Left Leaning Red Black Tree

```
from pyalgs.data_structures.commons.binary_search_tree import BinarySearchTree
bst = BinarySearchTree.create_red_black_tree()

bst.put("one", 1)
bst.put("two", 2)
bst.put("three", 3)
bst.put("six", 6)
bst.put("ten", 10)

print bst.get("one"))
print bst.contains_key("two")

for key in bst.keys():
    print(key)

print bst.size()
print bst.is_empty()

bst.delete("one")
```

Symbol Table using Hashed Map

```
from pyalgs.data_structures.commons.hashed_map import HashedMap
map = HashedMap.create()

map.put("one", 1)
map.put("two", 2)
map.put("three", 3)
map.put("six", 6)
map.put("ten", 10)

print map.get("one"))
print map.contains_key("two")

for key in map.keys():
    print(key)

print map.size()
print map.is_empty()

map.delete("one")
```

Symbol Table using Hashed Set

```
from pyalgs.data_structures.commons.hashed_set import HashedSet
set = HashedSet.create()

set.add("one")
set.add("two")
set.add("three")
set.add("six")
set.add("ten")

print set.contains("two")

for key in set.iterate():
    print(key)

print set.size()
print set.is_empty()

set.delete("one")
```

Graph

Undirected Graph

```
from pyalgs.data_structures.graphs.graph import Graph
G = Graph(100)

G.add_edge(1, 2)
G.add_edge(1, 3)

print([i for i in G.adj(1)])
```

```
print([i for i in G.adj(2)])
print([i for i in G.adj(3)])

print(G.vertex_count())
```

Directed Graph

```
from pyalgs.data_structures.graphs.graph import Digraph
G = Digraph(100)

G.add_edge(1, 2)
G.add_edge(1, 3)

print([i for i in G.adj(1)])
print([i for i in G.adj(2)])
print([i for i in G.adj(3)])

print(G.vertex_count())
```

Edge Weighted Graph

```
from pyalgs.data_structures.graphs.graph import EdgeWeightGraph
def create_edge_weighted_graph():
    g = EdgeWeightedGraph(8)
    g.add_edge(Edge(0, 7, 0.16))
    g.add_edge(Edge(2, 3, 0.17))
    g.add_edge(Edge(1, 7, 0.19))
    g.add_edge(Edge(0, 2, 0.26))
    g.add_edge(Edge(5, 7, 0.28))

    print([edge for edge in G.adj(3)])

    print(G.vertex_count())
    print(', '.join([edge for edge in G.edges()]))
    return g
```

Search Tries

Symbol Table using R-ways Search Tries

```
from pyalgs.data_structures.strings.search_tries import RWaySearchTries
st = RWaySearchTries()

st.put("one", 1)
st.put("two", 2)
st.put("three", 3)
st.put("six", 6)
st.put("ten", 10)

for key in st.keys():
    print(key)
```

```
print st.get("one"))
print st.contains_key("two")

print st.size()
print st.is_empty()

st.delete("one")

for key in st.keys_with_prefix('t'):
    print(key)
```

Symbol Table using Ternary Search Tries

```
from pyalgs.data_structures.strings.search_tries import TernarySearchTries
st = TernarySearchTries()

st.put("one", 1)
st.put("two", 2)
st.put("three", 3)
st.put("six", 6)
st.put("ten", 10)

for key in st.keys():
    print(key)

print st.get("one"))
print st.contains_key("two")

print st.size()
print st.is_empty()

st.delete("one")

for key in st.keys_with_prefix('t'):
    print(key)
```

Algorithms

Common Algorithms

Union Find

```
from pyalgs.algorithms.commons.union_find import UnionFind

uf = UnionFind.create(10)

uf.union(1, 3)
uf.union(2, 4)
uf.union(1, 5)

print(uf.connected(1, 3))
print(uf.connected(3, 5))
```

```
print(uf.connected(1, 2))
print(uf.connected(1, 4))
```

Sorting

The sorting algorithms sort an array in ascending order

Selection Sort

```
from pyalgs.algorithms.commons.sorting import SelectionSort

a = [4, 2, 1]
SelectionSort.sort(a)
print(a)
```

Insertion Sort

```
from pyalgs.algorithms.commons.sorting import InsertionSort

a = [4, 2, 1]
InsertionSort.sort(a)
print(a)
```

Shell Sort

```
from pyalgs.algorithms.commons.sorting import ShellSort

a = [4, 2, 1, 23, 4, 5, 6, 7, 8, 9, 20, 11, 13, 34, 66]
ShellSort.sort(a)
print(a)
```

Merge Sort

```
from pyalgs.algorithms.commons.sorting import MergeSort

a = [4, 2, 1, 23, 4, 5, 6, 7, 8, 9, 20, 11, 13, 34, 66]
MergeSort.sort(a)
print(a)
```

Quick Sort

```
from pyalgs.algorithms.commons.sorting import QuickSort

a = [4, 2, 1, 23, 4, 5, 6, 7, 8, 9, 20, 11, 13, 34, 66]
QuickSort.sort(a)
print(a)
```

3-Ways Quick Sort

```
from pyalgs.algorithms.commons.sorting import ThreeWayQuickSort

a = [4, 2, 1, 23, 4, 5, 6, 7, 8, 9, 20, 11, 13, 34, 66]
ThreeWayQuickSort.sort(a)
print(a)
```

Heap Sort

```
from pyalgs.algorithms.commons.sorting import HeapSort

a = [4, 2, 1, 23, 4, 5, 6, 7, 8, 9, 20, 11, 13, 34, 66]
HeapSort.sort(a)
print(a)
```

Shuffling

Knuth Shuffle

```
from pyalgs.algorithms.commons.shuffling import KnuthShuffle

a = [1, 2, 13, 22, 123]
KnuthShuffle.shuffle(a)
print(a)
```

Searching

Binary Selection

```
from pyalgs.algorithms.commons.selecting import BinarySelection
from pyalgs.algorithms.commons.util import is_sorted


a = [1, 2, 13, 22, 123]
assert is_sorted(a)
print BinarySelection.index_of(a, 13)
```

Graphs

Create Graphs

Create Undirected Graph

```
from pyalgs.data_structures.graphs.graph import Graph
def create_graph():
    G = Graph(100)
```

```
    G.add_edge(1, 2)
    G.add_edge(1, 3)

    print([i for i in G.adj(1)])
    print([i for i in G.adj(2)])
    print([i for i in G.adj(3)])

    print(G.vertex_count())
```

Create Directed Graph

```
from pyalgs.data_structures.graphs.graph import Digraph
def create_digraph():
    G = Digraph(100)

    G.add_edge(1, 2)
    G.add_edge(1, 3)

    print([i for i in G.adj(1)])
    print([i for i in G.adj(2)])
    print([i for i in G.adj(3)])

    print(G.vertex_count())
```

Edge Weighted Graph

```
from pyalgs.data_structures.graphs.graph import EdgeWeightGraph, Edge
def create_edge_weighted_graph():
    g = EdgeWeightedGraph(8)
    g.add_edge(Edge(0, 7, 0.16))
    g.add_edge(Edge(2, 3, 0.17))
    g.add_edge(Edge(1, 7, 0.19))
    g.add_edge(Edge(0, 2, 0.26))
    g.add_edge(Edge(5, 7, 0.28))

    print([edge for edge in G.adj(3)])

    print(G.vertex_count())
    print(', '.join([edge for edge in G.edges()]))
    return g
```

Directed Edge Weighted Graph

```
from pyalgs.data_structures.graphs.graph import DirectedEdgeWeightedGraph, Edge
def create_edge_weighted_digraph():
    g = DirectedEdgeWeightedGraph(8)

    g.add_edge(
        Edge(0, 1, 5.0))
    g.add_edge(
        Edge(0, 4, 9.0))
```

```
    g.add_edge(
        Edge(0, 7, 8.0))
    g.add_edge(
        Edge(1, 2, 12.0))
    return g
```

Flow Network (for max-flow min-cut problem)

```
from pyalgs.data_structures.graphs.graph import FlowNetwork, FlowEdge
def create_flow_network():
g = FlowNetwork(8)
g.add_edge(FlowEdge(0, 1, 10))
g.add_edge(FlowEdge(0, 2, 5))
g.add_edge(FlowEdge(0, 3, 15))
g.add_edge(FlowEdge(1, 4, 9))
g.add_edge(FlowEdge(1, 5, 15))
g.add_edge(FlowEdge(1, 2, 4))
g.add_edge(FlowEdge(2, 5, 8))
g.add_edge(FlowEdge(2, 3, 4))
g.add_edge(FlowEdge(3, 6, 16))
g.add_edge(FlowEdge(4, 5, 15))
g.add_edge(FlowEdge(4, 7, 10))
g.add_edge(FlowEdge(5, 7, 10))
g.add_edge(FlowEdge(5, 6, 15))
g.add_edge(FlowEdge(6, 2, 6))
g.add_edge(FlowEdge(6, 7, 10))

return g
```

Graph Search

Depth First Search

```
from pyalgs.algorithms.graphs.search import DepthFirstSearch
g = create_graph()
s = 0
dfs = DepthFirstSearch(g, s)

for v in range(1, g.vertex_count()):
    if dfs.hasPathTo(v):
        print(str(s) + ' is connected to ' + str(v))
        print('path is ' + ' => '.join([str(i) for i in dfs.pathTo(v)]))
```

Breadth First Search

```
from pyalgs.algorithms.graphs.search import BreadthFirstSearch
g = create_graph()
s = 0
dfs = BreadthFirstSearch(g, s)

for v in range(1, g.vertex_count()):
```

```
    if dfs.hasPathTo(v):
        print(str(s) + ' is connected to ' + str(v))
        print('path is ' + ' => '.join([str(i) for i in dfs.pathTo(v)]))
```

Graph Connectivity

Connected Components for undirected graph

```
from pyalgs.algorithms.graphs.connectivity import ConnectedComponents
G = create_graph()

cc = ConnectedComponents(G)
print('connected component count: ' + str(cc.count()))


for v in range(G.vertex_count()):
    print('id[' + str(v) + ']: ' + str(cc.id(v)))
for v in range(G.vertex_count()):
    r = randint(0, G.vertex_count()-1)
    if cc.connected(v, r):
        print(str(v) + ' is connected to ' + str(r))
```

Strongly Connected Components for directed graph

```
from pyalgs.algorithms.graphs.connectivity import StronglyConnectedComponents
G = create_graph()

cc = StronglyConnectedComponents(G)
print('strongly connected component count: ' + str(cc.count()))


for v in range(G.vertex_count()):
    print('id[' + str(v) + ']: ' + str(cc.id(v)))
for v in range(G.vertex_count()):
    r = randint(0, G.vertex_count()-1)
    if cc.connected(v, r):
        print(str(v) + ' is connected to ' + str(r))
```

Topological Sort

```
from pyalgs.algorithms.graphs.topological_sort import DepthFirstOrder
G = create_graph()
topological_sort = DepthFirstOrder(G)
print(' => '.join([str(i) for i in topological_sort.postOrder()]))
```

Cyclic Graph Detection

Directed Cycle Detection

```
from pyalgs.algorithms.graphs.directed_cycle import DirectedCycle
dag = create_dag()
    dc = DirectedCycle(dag)
    assertFalse(dc.hasCycle())
```

Minimum Spanning Tree

Minimum Spanning Tree (Kruskal)

```
from pyalgs.algorithms.graphs.minimum_spanning_trees import KruskalMST
g = create_edge_weighted_graph()
mst = KruskalMST(g)

tree = mst.spanning_tree()

for e in tree:
    print(e)
```

Minimum Spanning Tree (Lazy Prim)

```
from pyalgs.algorithms.graphs.minimum_spanning_trees import LazyPrimMST
g = create_edge_weighted_graph()
mst = LazyPrimMST(g)

tree = mst.spanning_tree()

for e in tree:
    print(e)
```

Shortest Path

Dijkstra

```
from pyalgs.algorithms.graphs.shortest_path import DijkstraShortestPath
g = create_edge_weighted_digraph()
s = 0
dijkstra = DijkstraShortestPath(g, s)
for v in range(1, g.vertex_count()):
    if dijkstra.hasPathTo(v):
        print(str(s) + ' is connected to ' + str(v))
        print('path is ' + ' .. '.join([str(i) for i in dijkstra.shortestPathTo(v)]))
        print('path length is ' + str(dijkstra.path_length_to(v)))
```

Shortest Path (Topological Sort)

```
from pyalgs.algorithms.graphs.shortest_path import TopologicalSortShortestPath
g = create_edge_weighted_digraph()
assert not DirectedCycle(g).hasCycle()
s = 0
dijkstra = TopologicalSortShortestPath(g, s)
for v in range(1, g.vertex_count()):
    if dijkstra.hasPathTo(v):
        print(str(s) + ' is connected to ' + str(v))
        print('shortest path is ' + ' .. '.join([str(i) for i in dijkstra.
↪shortestPathTo(v)]))
        print('path length is ' + str(dijkstra.path_length_to(v)))
```

Shortest Path (Bellman-Ford for positive and negative edge graph)

```
from pyalgs.algorithms.graphs.shortest_path import BellmanFordShortestPath
from pyalgs.algorithms.graphs.directed_cycle import DirectedCycle
g = create_edge_weighted_digraph()
s = 0
dijkstra = BellmanFordShortestPath(g, s)
for v in range(1, g.vertex_count()):
    if dijkstra.hasPathTo(v):
        print(str(s) + ' is connected to ' + str(v))
        print('shortest path is ' + ' .. '.join([str(i) for i in dijkstra.
↪shortestPathTo(v)]))
        print('path length is ' + str(dijkstra.path_length_to(v)))
```

Max-Flow-Min-Cut

MaxFlow MinCut (Ford-Fulkerson)

```
from pyalgs.algorithms.graphs.max_flow import FordFulkersonMaxFlow
network = create_flow_network()
ff = FordFulkersonMaxFlow(network, 0, 7)
print('max-flow: '+str(ff.max_flow_value()))
```

Strings

String Sorting

LSD Radix Sort

```
from pyalgs.algorithms.strings.sorting import LSD
LSD.sort(['good', 'cool', 'done', 'come'])
```

MSD Radix Sort

```
from pyalgs.algorithms.strings.sorting import MSD
words = 'more details are provided in the docs for implementation, complexities and␣
↪further info'.split(' ')
print(words)
MSD.sort(words)
print(words)
```

Sort (3-Ways String Quick Sort)

```
from pyalgs.algorithms.strings.sorting import String3WayQuickSort
words = 'more details are provided in the docs for implementation, complexities and␣
↪further info'.split(' ')
print(words)
String3WayQuickSort.sort(words)
print(words)
```

Substring Search

Substring Search (Brute force)

```
from pyalgs.algorithms.strings.substring_search import BruteForceSubstringSearch
ss = BruteForceSubstringSearch('find')
print(ss.search_in('I can find it here'))
print(ss.search_in('It is not here'))
```

Substring Search (Rabin Karp)

```
from pyalgs.algorithms.strings.substring_search import RabinKarp
ss = RabinKarp('find')
print(ss.search_in('I can find it here'))
print(ss.search_in('It is not here'))
```

Substring Search (Boyer Moore)

```
from pyalgs.algorithms.strings.substring_search import BoyerMoore
ss = BoyerMoore('find')
print(ss.search_in('I can find it here'))
print(ss.search_in('It is not here'))
```

Substring Search (Knuth Morris Pratt)

```
from pyalgs.algorithms.strings.substring_search import KnuthMorrisPratt
ss = KnuthMorrisPratt('find')
print(ss.search_in('I can find it here'))
print(ss.search_in('It is not here'))
```

Longest Repeated Substring

```
from pyalgs.algorithms.strings.longest_repeated_substring import␣
↪LongestRepeatedSubstringSearch
start, len = LongestRepeatedSubstringSearch.lrs('Hello World', 'World Record')
print('Hello World'[start:(start+len+1)])
```