API Documentation

API Documentation

December 10, 2015

Contents

1 Package datk

1.1 Modules

- **core**: A Python Toolkit for Distributed Algorithms
  (Section 2, p. 5)
  - **algs** *(Section 3, p. 6)*
  - **distalgs** *(Section 4, p. 18)*
  - **imports** *(Section 5, p. 25)*
  - **networks** *(Section 6, p. 26)*
  - **tester** *(Section 7, p. 30)*
- **tests** *(Section 8, p. 31)*
  - **helpers**: Helper functions for tests in tests.py
    (Section 9, p. 32)
  - **networks_tests**: Network Test Suite
    (Section 10, p. 33)
  - **tests**: Algorithm Test Suite
    (Section 11, p. 34)

1.2 Variables

Name	Description
__package__	**Value:** `None`

2 Package datk.core

A Python Toolkit for Distributed Algorithms

Authors:

- Amin Manna <manna@mit.edu>
- Mayuri Sridhar <mayuri@mit.edu>

2.1 Modules

- **algs** *(Section 3, p. 6)*
- **distalgs** *(Section 4, p. 18)*
- **imports** *(Section 5, p. 25)*
- **networks** *(Section 6, p. 26)*
- **tester** *(Section 7, p. 30)*

2.2 Variables

Name	Description
__package__	**Value:** `None`

3 Module datk.core.algs

3.1 Variables

Name	Description
__package__	**Value:** `'datk.core'`

3.2 Class LCR

datk.core.distalgs.Algorithm

datk.core.distalgs.Synchronous_Algorithm

datk.core.algs.LCR

The LeLann, Chang and Roberts algorithm for Leader Election in a Synchronous Ring Network

Each Process sends its identifier around the ring. When a Process receives an incoming identifier, it compares that identifier to its own. If the incoming identifier is greater than its own, it keeps passing the identifier; if it is less than its own, it discards the incoming identifier; if it is equal to its own, the Process declares itself the leader.

Requires:

- Every process knows state['n'], the size of the network

Effects:

- Every process has state['status'] is 'leader' or 'non-leader'.
- Exactly one process has state['status'] is 'leader'

3.2.1 Methods

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.3 Class AsyncLCR

datk.core.distalgs.Algorithm

datk.core.distalgs.Asynchronous_Algorithm

datk.core.algs.AsyncLCR

The LeLann, Chang and Roberts algorithm for Leader Election in an Asynchronous Ring Network

Each Process sends its identifier around the ring. When a Process receives incoming identifier(s), it compares their largest to its own. If that incoming identifier is greater than its own, it keeps passing that identifier; if it is less than its own, it discards all the incoming identifiers; if it is equal to its own, the Process declares itself the leader. When a Process has declared itself Leader, it sends a Leader Declaration message around the ring, and halts As it goes around the ring, each other Process outputs 'non-leader', and halts.

Requires:

- Every process knows state['n'], the size of the network

Effects:

- Every process has state['status'] is 'leader' or 'non-leader'.
- Exactly one process has state['status'] is 'leader'

3.3.1 Methods

msgs_i(*self*, *p*, *verbose*=`False`)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *verbose*=`False`)

Overrides: datk.core.distalgs.Algorithm.trans_i

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

Inherited from datk.core.distalgs.Asynchronous_Algorithm(Section 4.8)

run(), run_process()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.4 Class FloodMax

datk.core.distalgs.Algorithm

datk.core.distalgs.Synchronous_Algorithm

datk.core.algs.FloodMax

UID flooding algorithm for Leader Election in a general network

Every process maintains a record of the maximum UID it has seen so far (initially its own). At each round, each process propagates this maximum on all of its outgoing edges. After diam rounds, if the maximum value seen is the process's own UID, the process elects itself the leader; otherwise, it is a non-leader.

Requires:

- Every process, p, has p.state["diam"] >= dist(p, q), forall q.

3.4.1 Methods

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*, *verbose*=`False`)

Overrides: datk.core.distalgs.Algorithm.trans_i

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.5 Class SynchBFS

```
datk.core.distalgs.Algorithm ─┐
                              │
datk.core.distalgs.Synchronous_Algorithm ─┐
                                          │
                                 datk.core.algs.SynchBFS
```

Constructs a BFS tree with the 'leader' Process at its root

At any point during execution, there is some set of processes that is "marked," initially just i0. Process i0 sends out a search message at round 1, to all of its outgoing neighbors. At any round, if an unmarked process receives a search message, it marks itself and chooses one of the processes from which the search has arrived as its parent. At the first round after a process gets marked, it sends a search message to all of its outgoing neighbors.

Requires:

- testLeaderElection

Effects:

- every Process has state['parent']. Leader has state['parent'] = None

3.5.1 Methods

is_i0(*self*, *p*)

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.6 Class SynchBFSAck

```
datk.core.distalgs.Algorithm ─┐
                              │
datk.core.distalgs.Synchronous_Algorithm ─┐
                                          │
                                          datk.core.algs.SynchBFSAck
```

Constructs a BFS tree with children pointers and the 'leader' Process at its root

Algorithm (Informal): At any point during execution, there is some set of processes that is "marked," initially just i0. Process i0 sends out a search message at round 1, to all of its outgoing neighbors. At any round, if an unmarked process receives a search message, it marks itself and chooses one of the processes from which the search arrived as its parent. At the first round after a process gets marked, it sends a search message to all of its outgoing neighbors, and an acknowledgement to its parent, so that nodes will also know their children.

Requires:

- testLeaderElection

Effects:

- Every process knows:
  - state['parent']. Leader has state['parent'] = None
  - state['childen']. Leaves have state['children'] = []

3.6.1 Methods

is_i0(*self*, *p*)

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.7 Class SynchConvergecast

```
datk.core.distalgs.Algorithm ─┐
                              │
datk.core.distalgs.Synchronous_Algorithm ─┐
                                          │
                                          datk.core.algs.SynchConvergecast
```

Known Subclasses: datk.core.algs.SynchConvergeHeight

The abstract superclass of a class of Synchronous Algorithms that propagate information from the leaves of a BFS tree to its root.

Requires:

- Every Process knows state['parent']

#TODO If Processes also know state['children'] ==> Reduced Communication Complexity.

3.7.1 Methods

is_root(*self*, *p*)

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

trans_root(*self*, *p*, *msgs*)

outpout_root(*self*, *p*)

initial_msg_to_parent(*self*, *p*)

trans_msg_to_parent(*self*, *p*, *msgs*)

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.8 Class AsynchConvergecast

datk.core.distalgs.Algorithm
datk.core.distalgs.Asynchronous_Algorithm
datk.core.algs.AsynchConvergecast

Known Subclasses: datk.core.algs.AsynchConvergeHeight

The abstract superclass of a class of Synchronous Algorithms that propagate information from the leaves of a BFS tree to its root.

Requires:

- Every Process knows state['parent'] and state['children']

3.8.1 Methods

is_root(*self*, *p*)

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

trans_root(*self*, *p*, *msgs*)

outpout_root(*self*, *p*)

initial_msg_to_parent(*self*, *p*)

trans_msg_to_parent(*self*, *p*, *msgs*)

Inherited from datk.core.distalgs.Asynchronous_Algorithm(Section 4.8)

run(), run_process()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.9 Class SynchConvergeHeight

datk.core.distalgs.Algorithm

datk.core.distalgs.Synchronous_Algorithm

datk.core.algs.SynchConvergecast

datk.core.algs.SynchConvergeHeight

Requires:

- BFS Tree

Effects:

- Root Process knows height of tree in state["height"]

3.9.1 Methods

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

initial_msg_to_parent(*self*, *p*)

Overrides: datk.core.algs.SynchConvergecast.initial_msg_to_parent

output_root(*self*, *p*)

trans_msg_to_parent(*self*, *p*, *msgs*)

Overrides: datk.core.algs.SynchConvergecast.trans_msg_to_parent

trans_root(*self*, *p*, *msgs*)

Overrides: datk.core.algs.SynchConvergecast.trans_root

Inherited from datk.core.algs.SynchConvergecast(Section 3.7)

is_root(), msgs_i(), outpout_root(), trans_i()

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.10 Class AsynchConvergeHeight

datk.core.distalgs.Algorithm

datk.core.distalgs.Asynchronous_Algorithm

datk.core.algs.AsynchConvergecast

datk.core.algs.AsynchConvergeHeight

Requires:

- BFS Tree

Effects:

- Root Process knows height of tree in state["height"]

3.10.1 Methods

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

initial_msg_to_parent(*self*, *p*)

Overrides: datk.core.algs.AsynchConvergecast.initial_msg_to_parent

output_root(*self*, *p*)

trans_msg_to_parent(*self*, *p*, *msgs*)

Overrides: datk.core.algs.AsynchConvergecast.trans_msg_to_parent

trans_root(*self*, *p*, *msgs*)

Overrides: datk.core.algs.AsynchConvergecast.trans_root

Inherited from datk.core.algs.AsynchConvergecast(Section 3.8)

is_root(), msgs_i(), outpout_root(), trans_i()

Inherited from datk.core.distalgs.Asynchronous_Algorithm(Section 4.8)

run(), run_process()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.11 Class SynchBroadcast

datk.core.distalgs.Algorithm

datk.core.distalgs.Synchronous_Algorithm

datk.core.algs.SynchBroadcast

Broadcasts a value stored in Process, p, to the BFS tree rooted at p

Requires:

- The attribute to be broadcasted must be specified in self.params['attr']
- BFS Tree with children pointers, where root node has state[self.params['attr']]

Effects:

- All Processes have state[self.params['attr']] := the original value of in state[self.params['attr']] of the root Process.

3.11.1 Methods

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.12 Class SynchLubyMIS

```
datk.core.distalgs.Algorithm ─┐
                              │
datk.core.distalgs.Synchronous_Algorithm ─┐
                                          │
                                          datk.core.algs.SynchLubyMIS
```

A randomized algorithm that constructs a Maximal Independent Set

The algorithm works in stages, each consisting of three rounds.

- Round 1: In the first round of a stage, the processes choose their respective vals and send them to their neighbors. By the end of round 1, when all the val messages have been received, the winners–that is, the processes in F–know who they are.
- Round 2: In the second round, the winners notify their neighbors. By the end of round 2, the losers–that is, the processes having neighbors in F–know who they are.
- Round 3: In the third round, each loser notifies its neighbors. Then all the involved processes–the winners, the losers, and the losers' neighbors– remove the appropriate nodes and edges from the graph. More precisely, this means the winners and losers discontinue participation after this stage, and the losers' neighbors remove all the edges that are incident on the newly removed nodes.

Requires:

- Every process knows state['n'], the size of the network

Effect:

- Every process knows state['MIS']. A boolean representing whether it is a member of the Maximal Independent Set found by Luby's algorithm.

3.12.1 Methods

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

4 Module datk.core.distalgs

4.1 Variables

Name	Description
__package__	**Value:** `'datk.core'`

4.2 Class Message

Known Subclasses: datk.core.algs.AsyncLCR.Leader_Declaration, datk.core.algs.SynchBFS.Search, datk.core.algs.SynchBFSAck.AckParent, datk.core.algs.SynchBFSAck.Search

4.2.1 Methods

__init__(*self*, *algorithm*, *content*=`None`)

__str__(*self*)

4.3 Class Process

A computing element located at a node of a network graph. Processes are identical except for their UID

4.3.1 Methods

__init__(*self*, *UID*, *state*=`None`, *in_nbrs*=`[]`, *out_nbrs*=`[]`)

link_to(*self*, *new_out_nbr*)

Adds a new outgoing neighbor of the Process

bi_link(*self*, *nbr*)

output(*self*, *key*, *val*, *silent*=`False`)

send_to_all_neighbors(*self*, *msg*)

send_msg(*self*, *msg*, *out_nbrs*=`None`)

get_msgs(*self*, *algorithm*, *in_nbrs*=`None`)

add(*self*, *algorithm*)

terminate(*self*, *algorithm*)

__str__(*self*)

__repr__(*self*)

4.4 Class Network

Known Subclasses: datk.core.networks.Bidirectional_Line, datk.core.networks.Bidirectional_Ring, datk.core.networks.Complete_Graph, datk.core.networks.Random_Line_Network, datk.core.networks.Unidi
datk.core.networks.Unidirectional_Ring

A collection of Processes that know n, the # of processes in the network.

4.4.1 Methods

__getitem__(*self*, *i*)

__init__(*self*, *n*, *index_to_UID*=`None`)

Creates a network of n disconnected Processes, with random distinct UIDs, or as specified by the index_to_UID function

__iter__(*self*)

__len__(*self*)

__repr__(*self*)

add(*self*, *algorithm*)

clone(*self*)

draw(*self*)

index(*self*, *p*)

run(*self*, *algorithm*)

state(*self*)

4.5 Class Algorithm

Known Subclasses: datk.core.distalgs.Asynchronous_Algorithm, datk.core.distalgs.Chain, datk.core.distalgs.Synchronous_Algorithm

Abstract superclass for a distributed algorithm.

4.5.1 Methods

__init__(*self*, *network*=`None`, *params*={`'draw': False, 'silent': True`}, *name*=`None`)

Parameters

`network:`	[Optional] network. If specified, algorithm is immediately executed on network.
`params:`	[Optional] runtime parameters.
`name:`	[Optional] name of the Algorithm instance. Defaults to class name.

msgs_i(*self*, *p*)

trans_i(*self*, *p*, *msgs*)

halt_i(*self*, *p*)

cleanup_i(*self*, *p*)

cleanup(*self*)

__call__(*self*, *network*, *params*={})

run(*self*, *network*, *params*={})

halt(*self*)

count_msg(*self*, *message_count*)

set(*self*, *process*, *state*, *value*)

increment(*self*, *process*, *state*, *inc*=1)

has(*self*, *process*, *state*)

get(*self*, *process*, *state*)

delete(*self*, *process*, *state*)

4.6 Class Synchronous_Algorithm

```
datk.core.distalgs.Algorithm ─┐
                              datk.core.distalgs.Synchronous_Algorithm
```

Known Subclasses: datk.core.distalgs.Compose, datk.core.distalgs.Do_Nothing, datk.core.algs.FloodMax
datk.core.algs.LCR, datk.core.algs.SynchBFS, datk.core.algs.SynchBFSAck, datk.core.algs.SynchBroadcast
datk.core.algs.SynchConvergecast, datk.core.algs.SynchLubyMIS

We assume that Processes take steps simultaneously, that is, that execution proceeds in synchronous rounds.

4.6.1 Methods

execute(*self*)

msgs(*self*)

round(*self*)

run(*self*, *network*, *params*={})

Overrides: datk.core.distalgs.Algorithm.run

trans(*self*)

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(),

halt_i(), has(), increment(), msgs_i(), set(), trans_i()

4.7 Class Do_Nothing

datk.core.distalgs.Algorithm ┐

datk.core.distalgs.Synchronous_Algorithm ┐

datk.core.distalgs.Do_Nothing

4.7.1 Methods

trans_i(*self*, *p*, *messages*)

Overrides: datk.core.distalgs.Algorithm.trans_i

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), msgs_i(), set()

4.8 Class Asynchronous_Algorithm

datk.core.distalgs.Algorithm ┐

datk.core.distalgs.Asynchronous_Algorithm

Known Subclasses: datk.core.algs.AsyncLCR, datk.core.algs.AsynchConvergecast

We assume that the separate Processes take steps in an arbitrary order, at arbitrary relative speeds.

4.8.1 Methods

run(*self*, *network*, *params*={})

Overrides: datk.core.distalgs.Algorithm.run

run_process(*self*, *process*)

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), msgs_i(), set(), trans_i()

4.9 Class Compose

```
datk.core.distalgs.Algorithm ─┐
                              │
datk.core.distalgs.Synchronous_Algorithm ─┐
                                          │
                                  datk.core.distalgs.Compose
```

A Synchonous_Algorithm that is the composition of two synchronous algorithms running in parallel.

4.9.1 Methods

__init__(*self*, *A*, *B*, *name*=`None`, *params*={`'draw': False, 'silent': True`})

Parameters

`network:`	[Optional] network. If specified, algorithm is immediately executed on network.
`params:`	[Optional] runtime parameters.
`name:`	[Optional] name of the Algorithm instance. Defaults to class name.

Overrides: datk.core.distalgs.Algorithm.__init__

msgs_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.msgs_i

trans_i(*self*, *p*, *msgs*)

Overrides: datk.core.distalgs.Algorithm.trans_i

halt_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.halt_i

cleanup_i(*self*, *p*)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

run(*self*, *network*, *params*={})

Overrides: datk.core.distalgs.Algorithm.run

__repr__(*self*)

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), cleanup(), count_msg(), delete(), get(), halt(), has(), increment(), set()

4.10 Class Chain

datk.core.distalgs.Algorithm ─┐
datk.core.distalgs.Chain

An Algorithm that is the result of sequentially running two algorithms

4.10.1 Methods

__init__(*self*, *A*, *B*, *name*=`None`, *params*={`'draw': False, 'silent': False`})

Parameters

`network`:	[Optional] network. If specified, algorithm is immediately executed on network.
`params`:	[Optional] runtime parameters.
`name`:	[Optional] name of the Algorithm instance. Defaults to class name.

Overrides: datk.core.distalgs.Algorithm.__init__

run(*self*, *network*, *params*={})

Overrides: datk.core.distalgs.Algorithm.run

__repr__(*self*)

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), msgs_i(), set(), trans_i()

5 Module datk.core.imports

5.1 Variables

Name	Description
TIMEOUT	**Value:** `5`
__package__	**Value:** `'datk.core'`
failed_tests	**Value:** `set([])`
lock	**Value:** `Lock()`
num_tests	**Value:** `0`

6 Module datk.core.networks

6.1 Variables

Name	Description
__package__	**Value:** `'datk.core'`

6.2 Class Unidirectional_Ring

datk.core.distalgs.Network ─┐
datk.core.networks.Unidirectional_Ring

A Network of n Processes arranged in a ring. Each edge is directed from a Process to its clockwise neighbor, that is, messages can only be sent in a clockwise direction.

6.2.1 Methods

__init__(*self*, *n*, *index_to_UID*=`None`)

Creates a network of n disconnected Processes, with random distinct UIDs, or as specified by the index_to_UID function

Overrides: datk.core.distalgs.Network.__init__ extit(inherited documentation)

Inherited from datk.core.distalgs.Network(Section 4.4)

__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()

6.3 Class Bidirectional_Ring

datk.core.distalgs.Network ─┐
datk.core.networks.Bidirectional_Ring

A Network of n Processes arranged in a ring. Each edge between a Process and its neighbor is undirected, that is, messages can be sent in both the clockwise and the counterclockwise directions.

6.3.1 Methods

__init__(*self*, *n*, *index_to_UID*=`None`)

Creates a network of n disconnected Processes, with random distinct UIDs, or as specified by the index_to_UID function

Overrides: datk.core.distalgs.Network.__init__ extit(inherited documentation)

Inherited from datk.core.distalgs.Network(Section 4.4)

__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()

6.4 Class Unidirectional_Line

datk.core.distalgs.Network

datk.core.networks.Unidirectional_Line

A Network of n Processes arranged in a line. Each edge is directed from a Process to its clockwise neighbor, that is, messages can only be sent in a clockwise direction.

6.4.1 Methods

__init__(*self*, *n*, *index_to_UID*=`None`)

Creates a network of n disconnected Processes, with random distinct UIDs, or as specified by the index_to_UID function

Overrides: datk.core.distalgs.Network.__init__ extit(inherited documentation)

Inherited from datk.core.distalgs.Network(Section 4.4)

__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()

6.5 Class Bidirectional_Line

datk.core.distalgs.Network

datk.core.networks.Bidirectional_Line

A Network of n Processes arranged in a line. Each edge between a Process and its neighbor is undirected, that is, messages can be sent in both the clockwise and the counterclockwise directions.

6.5.1 Methods

__init__(*self*, *n*, *index_to_UID*=`None`)

Creates a network of n disconnected Processes, with random distinct UIDs, or as specified by the index_to_UID function

Overrides: datk.core.distalgs.Network.__init__ extit(inherited documentation)

Inherited from datk.core.distalgs.Network(Section 4.4)

__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()

6.6 Class Complete_Graph

datk.core.distalgs.Network ─┐
datk.core.networks.Complete_Graph

A Network of n Processes arranged at the vertices of a Complete undirected graph of size n.

6.6.1 Methods

__init__(*self*, *n*, *index_to_UID*=`None`)

Creates a network of n disconnected Processes, with random distinct UIDs, or as specified by the index_to_UID function

Overrides: datk.core.distalgs.Network.__init__ extit(inherited documentation)

Inherited from datk.core.distalgs.Network(Section 4.4)

__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()

6.7 Class Random_Line_Network

datk.core.distalgs.Network ─┐
datk.core.networks.Random_Line_Network

A Network of n processes arranged randomly at the vertices of a connected undirected line graph of size n. Additional pairs of vertices are connected at random with a probability that is inversely proportional to the difference in their positions on the line.

For example, the Process at index 3 is guaranteed to be connected to the Process at index 4, and is more likely to be connected to the Process at index 5 than to the Process at index 8.

6.7.1 Methods

__init__(*self*, *n*, *sparsity*=1)

sparsity = 0 –> a Complete_Graph(n) sparsity = infinity –> a Bidirectional_Line(n)

Overrides: datk.core.distalgs.Network.__init__

Inherited from datk.core.distalgs.Network(Section 4.4)

__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()

7 Module datk.core.tester

7.1 Functions

test(*f*=`None`, *timeout*=`5`, *main_thread*=`False`, *test*=`True`)

print_with_underline(*text*)

summarize()

7.2 Variables

Name	Description
TIMEOUT	**Value:** `5`
lock	**Value:** `Lock()`
num_tests	**Value:** `0`
failed_tests	**Value:** `set([])`
__package__	**Value:** `'datk.core'`

8 Package datk.tests

8.1 Modules

- **helpers**: Helper functions for tests in tests.py
  (Section 9, p. 32)
- **networks_tests**: Network Test Suite
  (Section 10, p. 33)
- **tests**: Algorithm Test Suite
  (Section 11, p. 34)

8.2 Variables

Name	Description
__package__	**Value:** `None`

9 Module datk.tests.helpers

Helper functions for tests in tests.py

9.1 Functions

testLeaderElection(*network*, *isLeader*=`<function <lambda> at 0x1d1a430>`, *isNonleader*=`<function <lambda> at 0x1d1a470>`)

Asserts that exactly one Process is Leader, and all other processes are Non-Leader

testBroadcast(*network*, *attr*)

Asserts that p.state[attr] is identical for all processes p

testBFS(*network*)

Asserts that every Process, p, knows 'parent', and there exists exactly one Process where 'parent' is None

testBFSWithChildren(*network*)

Asserts that every Process, p, knows 'parent' and 'children', and there exists exactly one Process where 'parent' is None

testLubyMIS(*network*)

Asserts that every process knows a boolean value, 'MIS', and that the Processes where 'MIS' is True form a set that is both independent and maximal.

9.2 Variables

Name	Description
__package__	**Value:** `'datk.tests'`

10 Module datk.tests.networks_tests

Network Test Suite

Tests Netwoks defined in networks.py by visual inspection

10.1 Variables

Name	Description
DRAW_RANDOM	**Value:** `None`
DRAW_HUGE_RANDOM	**Value:** `None`
DRAW_UNI_RING	**Value:** `None`
DRAW_BI_RING	**Value:** `None`
DRAW_COMPLETE_GR-APH	**Value:** `None`
DRAW_UNI_LINE	**Value:** `None`
DRAW_BI_LINE	**Value:** `None`
TIMEOUT	**Value:** `5`
__package__	**Value:** `'datk.tests'`
failed_tests	**Value:** `set([])`
lock	**Value:** `Lock()`
num_tests	**Value:** `0`

11 Module datk.tests.tests

Algorithm Test Suite

Tests algorithms defined in algs.py

11.1 Functions

configure_ipython()

Convenient helper function to determine if environment is ipython. Note that drawing is only safe in ipython qtconsole with matplotlib inline If environment is IPython, returns True and configures IPython. Else returns False.

LCR_UNI_RING()

LCR_BI_RING()

ASYNC_LCR_UNI_RING()

ASYNC_LCR_BI_RING()

FLOODMAX_UNI_RING()

FLOODMAX_BI_RING()

FLOODMAX_BI_LINE()

FLOODMAX_COMPLETE_GRAPH()

FLOODMAX_RANDOM_GRAPH()

SYNCH_BFS()

SYNCH_BFS_ACK()

SYNCH_CONVERGE_HEIGHT()

SYNCH_BROADCAST_HEIGHT()

ASYNCH_BROADCAST_HEIGHT()

send_receive_msgs()

SYNCH_DO_NOTHING()

COMPOSE_SYNCH_LCR_AND_DO_NOTHING()

COMPOSE_SYNCH_LCR()

CHAIN_BROADCAST_HEIGHT()

SYNCH_LUBY_MIS_BI_RING()

SYNCH_LUBY_MIS()

11.2 Variables

Name	Description
in_ipython	**Value:** `False`
test_params	**Value:** `{'draw': False, 'silent': True}`
TIMEOUT	**Value:** `5`
__package__	**Value:** `'datk.tests'`
failed_tests	**Value:** `set([])`
lock	**Value:** `Lock()`
num_tests	**Value:** `0`

Index