PySkipList Documentation

Release 1.0.0

Geert Jansen

July 11, 2015

class pyskiplist.**SkipList**
An indexable skip list.

A SkipList provides an ordered sequence of key-value pairs. The list is always sorted on key and supports O(1) forward iteration. It has O(log N) time complexity for key lookup, pair insertion and pair removal anywhere in the list. The list also supports O(log N) element access by position.

The keys of all pairs you add to the skiplist must be be comparable against each other, and define the < and <= operators.

level
The current level of the skip list.

insert(*key*, *value*)
Insert a key-value pair in the list.

The pair is inserted at the correct location so that the list remains sorted on *key*. If a pair with the same key is already in the list, then the pair is appended after all other pairs with that key.

replace(*key*, *value*)
Replace the value of the first key-value pair with key *key*.

If the key was not found, the pair is inserted.

clear()
Remove all key-value pairs.

__len__()
Return the number of pairs in the list.

items(*start=None*, *stop=None*)
Return an iterator yielding pairs.

If *start* is specified, iteration starts at the first pair with a key that is larger than or equal to *start*. If not specified, iteration starts at the first pair in the list.

If *stop* is specified, iteration stops at the last pair that is smaller than *stop*. If not specified, iteration end with the last pair in the list.

__iter__(*start=None*, *stop=None*)
Return an iterator yielding pairs.

If *start* is specified, iteration starts at the first pair with a key that is larger than or equal to *start*. If not specified, iteration starts at the first pair in the list.

If *stop* is specified, iteration stops at the last pair that is smaller than *stop*. If not specified, iteration end with the last pair in the list.

keys(*start=None*, *stop=None*)
Like *items()* but returns only the keys.

values(*start=None*, *stop=None*)
Like *items()* but returns only the values.

popitem()
Removes the first key-value pair and return it.

This method raises a KeyError if the list is empty.

search(*key*, *default=None*)
Find the first key-value pair with key *key* and return its value.

If the key was not found, return *default*. If no default was provided, return None. This method never raises a KeyError.

remove(*key*)
Remove the first key-value pair with key *key*.

If the key was not found, a `KeyError` is raised.

pop(*key*, *default=<object object>*)
Remove the first key-value pair with key *key*.

If a pair was removed, return its value. Otherwise if *default* was provided, return *default*. Otherwise a `KeyError` is raised.

__contains__(*key*)
Return whether *key* is contained in the list.

index(*key*, *default=<object object>*)
Find the first key-value pair with key *key* and return its position.

If the key is not found, return *default*. If default was not provided, raise a `KeyError`

count(*key*)
Return the number of pairs with key *key*.

__getitem__(*pos*)
Return a pair by its position.

If *pos* is a slice, then return a generator that yields pairs as specified by the slice.

__delitem__(*pos*)
Delete a pair by its position.

__setitem__(*pos*, *value*)
Set a value by its position.

class pyskiplist.**Node**(*value=None*)
Base node class for *dllist*.

You can create a custom node with extra attributes by inheriting from this class. When you do this you need to set the `'__slots__'` attribute to include your custom attributes, and include `'_prev'` and `'_next'` also.

class pyskiplist.**dllist**
A doubly linked list.

first
The first node in the list.

last
The last node in the list.

__len__()
Return the number of nodes in this list.

remove(*node*)
Remove a node from the list.

The *node* argument must be a node that was previously inserted in the list

insert(*node*, *before=None*)
Insert a new node in the list.

The *node* argument must be a *Node* instance.

If *before* is not provided (the default), the node is appended at the end of the list. If *before* is provided, it must be a *Node* instance that is already part of this list, and the node is inserted before this node.

To insert at the start of the list, set *before* to *first*.

__iter__()
Return an iterator/generator that yields all nodes.

Note: it is safe to remove the current node while iterating but you should not remove the next one.

Symbols

C

D

F

I

K

L

N

P

R

S

V