Varun Rajgopal
CS 562
Dr. Alex Groce

CS 562: Applied Software Engineering
Project Proposal

Python Library:

I will use TSTL to test a python library "bintrees 2.0.2". It is a package that provides Binary-, RedBlack- and AVL-Tress in python. The code is simple and easily extensible. It does not require any external libraries for its installation. The classes implemented are much slower than the built-in dict class. All the iterators/ generators that yield data are in sorted key order. Trees can be uses as drop in replacement for dicts in most cases. The following are the trees written in python as part of Binary Tree package:

- Binary Tree – unbalanced binary tree
- AVLTree – balanced AVL- Tree
- RBTree – balanced Red-Black-Tree

Input:

All trees provide the same API and the pickle protocol is supported.

Following are the constructors of the binary trees:

- Tree() -> new empty tree;
- Tree(mapping) -> new tree initialized from a mapping (requires only an items() method)
- Tree(seq) -> new tree initialized from seq [(k1, v1), (k2, v2), ... (kn, vn)]

Following are some of the methods the binary trees use:

- __contains__(k) -> True if T has a key k, else False, O(log(n))
- __delitem__(y) <==> del T[y], del[s:e], O(log(n))
- __getitem__(y) <==> T[y], T[s:e], O(log(n))
- __iter__() <==> iter(T)
- __len__() <==> len(T), O(1)
- __max__() <==> max(T), get max item (k,v) of T, O(log(n))
- __min__() <==> min(T), get min item (k,v) of T, O(log(n))
- __and__(other) <==> T & other, intersection
- __or__(other) <==> T | other, union
- __sub__(other) <==> T - other, difference
- __xor__(other) <==> T ^ other, symmetric_difference
- __repr__() <==> repr(T)
- __setitem__(k, v) <==> T[k] = v, O(log(n))
- clear() -> None, remove all items from T, O(n)
- copy() -> a shallow copy of T, O(n*log(n))
- discard(k) -> None, remove k from T, if k is present, O(log(n))

Varun Rajgopal
CS 562
Dr. Alex Groce

- get(k[,d]) -> T[k] if k in T, else d, O(log(n))
- is_empty() -> True if len(T) == 0, O(1)
- items([reverse]) -> generator for (k, v) items of T, O(n)
- keys([reverse]) -> generator for keys of T, O(n)
- values([reverse]) -> generator for values of T, O(n)
- pop(k[,d]) -> v, remove specified key and return the corresponding value, O(log(n))
- pop_item() -> (k, v), remove and return some (key, value) pair as a 2-tuple, O(log(n)) (synonym popitem() exist)
- set_default(k[,d]) -> value, T.get(k, d), also set T[k]=d if k not in T, O(log(n)) (synonym setdefault() exist)
- update(E) -> None. Update T from dict/iterable E, O(E*log(n))
- foreach(f, [order]) -> visit all nodes of tree (0 = 'inorder', -1 = 'preorder' or +1 = 'postorder') and call f(k, v) for each node, O(n)
- iter_items(s, e[, reverse]) -> generator for (k, v) items of T for s <= key < e, O(n) remove_items(keys) -> None, remove items by keys, O(n)

Test:

I would like to test and ensure correctness of the heap methods for each of the trees. Following are the heap methods that are implemented as part of the library.

- max_item() -> get largest (key, value) pair of T, O(log(n))
- max_key() -> get largest key of T, O(log(n))
- min_item() -> get smallest (key, value) pair of T, O(log(n))
- min_key() -> get smallest key of T, O(log(n))
- pop_min() -> (k, v), remove item with minimum key, O(log(n))
- pop_max() -> (k, v), remove item with maximum key, O(log(n))
- nlargest(i[,pop]) -> get list of i largest items (k, v), O(i*log(n))
- nsmallest(i[,pop]) -> get list of i smallest items (k, v), O(i*log(n))

About the System:

- Bintrees can be found on bitbucket.org at http://bitbucket.org/mozman/bintrees