COMP 352 Fall Semester 2017

Assignment 4

Rui Zhao

40018813

Programming part:

Design report

In order to dynamically adjust the structure of the SmartAR ADT, I implement the SmartAR class, which will be implemented as a sequence (double-linked list) when the number of keys of the data is smaller than the threshold, and as hash table when it is bigger, basedon the threshold that has been set. These two different implementations are achieved by SmallSmartAR classe and BigSmartAR class.

Therefore, the implementation for functions in these two cases will be different, except function setThreshold and setKeyLength (which are implemented in the SmartAR class).

setThreshold(Threshold)

this.threshold ← Threshold

setKeyLength(Length)

this.keyLength ← Length

1) **When data size is small**

   When data size is small, SmartAR is implemented as a sequence using double-linked list as its underlying data type.

   Pseudo code of the methods

   Algorithm generate(n)

Input: a number n

Output: sequence with n random generated keys

Sequence s;

for i←0 to n-1

random(key)

s.addToLast(key)

Algorithm allKeys()

Input: Sequence s

Output: array of string A

While s.hasNext

A.add(s.next)

Return A

Algorithm add(key,value)

Input: a pair of data

Output: sequence with the pair added

Sequence s;

if key is not in the sequence then

sortedAdd this pair into sequence

else then

Node n is the node containing duplicate key

n.value.add(value);

Algorithm remove(key)

Input: a key

Output: sequence with the key removed

Sequence s;

if s.contains(key) then

Node n is the node containing the key

s.remove(n)

Algorithm getValue(key)

Input: a key

Output: value corresponding to the key

Sequence s;

if s.contains(key) then

Node n is the node containing the key

return n.value

Algorithm nextKey(key)

Input: a key

Output: successor of the key

Sequence s ← allKeys()

Search node n that contains the given key using binary search

return n.prev.key

Algorithm prevKey(key)

Input: a key

Output: predecessor of the key

Sequence s ← allKeys()

Search node n that contains the given key using binary search

return n.next.key

Algorithm previousCars(key)

Input: a key

Output: previous cars that are registered under the key

if s.contains(key) then

Node n is the node containing the key

return n.value

The underlying data type of my sequence is double-linked list. For which the complexity of sortedAdd is O(n^2) , and since the sequence is already sorted, allKey() function is O(n). Remove, getValue() and previousCars(), nextKey() and prevKey() will be O(n) (have to find the node) .

If array is used as the underlying data type instead (and the size of the array is enough for all the data), then add is O(1). allKey() uses merge sort which is also O(nlogn). After sorting the sequence, remove, getValue() and previousCars() will be O(logn) (binary search), nextKey() and prevKey() will be O(1) .

2) When data size is big

when data is big, I chose to use AVL tree as the implementation of our SmartAR ADT. Although hash table is very fast in terms of search, it is not continent to do sorting (or to retrieve the next and previous key). We have to use an extra array to store and sort the keys using in-place quick sort (which is usually the most fast sorting algorithm for large data). That's why I used AVL tree instead. As a result, add n elements and generating n random keys are O(nlogn), removing an element from the hash table are O(logn), getValue() and previousCars() are also O(logn). Getting next or previous key will be O(1). In terms of space complexity, we need the space for the AVL tree.

Pseudo code of the methods

Algorithm generate(n)

Input: a number n

Output: AVL tree T

for i←0 to n-1

T.add(random)

Algorithm allKeys()

Input: all keys from data

Output: sorted array A of all the keys

A ← array that stores all keys

While T.hasNext

A.add(T.next())

return A

Algorithm add(key,value)

Input: a pair of data

Output: updated AVL tree T

if key is not in the tree then

T.add(key)

else then

Node n is the node containing duplicate key

n.value.add(value);

Algorithm remove(key)

Input: a key

Output: updated AVL tree T

h.remove(key)

Algorithm getValue(key)

Input: a key

Output: value corresponding to the key

return T.find(key)

Algorithm nextKey(key)

Input: a key

Output: successor of the key

return T.next(key)

Algorithm prevKey(key)

Input: a key

Output: predecessor of the key

return T.previous(key)

Algorithm previousCars(key)

Input: a key

Output: previous cars that are registered under the key

Node n is the node containing the key

Return n.getElement;