An environment

Nariman Haidar

January 27, 2023

1 Introduction

In this assignment we are going to implement the map using two different techniques list and tree. Then we are going to do some benchmarks to construct a map of a number of elements and then measure the time it takes to perform an operation.And then compare the two implementations of the key-value store.

2 A map as a list:

In this section we are going to implement the small map that represent it as a list of key value tuples.

A list saves its elements in a sorted order, therefore for the list to be ordered, a new element needs to be added in the proper spot on the list. When we implementing a map with sorted lists, we avoid having to go through the full list for keys that have previously been used.

To do this map list so first we are going to define a public function EnvList, then define new to return an empty map.

Then define add(map, key, value) to return a map where an association of the key key and the data structure value has been added to the given map.

Then define lookup(map, key) to return either key, value. Then define the remove(key, map) to return a map where the association of the key key has been removed.

2.1 A map as a tree:

We used the list when we had a small map but when the map is large we are going to use the tree where the tree is another type of data structure for storing data is the tree structure which is also is better approach than list.

There are two types of elements in a tree: the exterior nodes, which are elements without children in the tree, and the inside nodes, which have sons.

The tree has the root where it is highest element in the tree and it is separated into two sections.

To implement the tree first we are going to define the public function EnvTree then define the new() then define add to add values to the left and to the right into tree and we used the recursion function to add all key-value pair to the empty tree, then in the second add we are going to replace the key if it has been found and in the third add we are going to return a tree that looks like the one we have but where the left branch has been updated and then do the same thing to the right.

Then we are going to implement lookup function that is going to look up the key and here we are not building a new tree just looking up and then returning the found key-value pair or nil if not found.

Then we are going to implement the remove function. This function is going to find the key to be removed first, then replacing it with the leftmost key-value combination in the right branch.

2.2 Benchmark:

Now we are going to do some benchmarks to construct a map of a number of elements and then measure the time it takes to perform an operation add, look up and remove.

n(us)	**add**	**look up**	**remove**
16	0.11	0.11	0.19
32	0.12	0.09	0.16
64	0.23	0.11	0.21
128	0.25	0.09	0.31
256	0.5	0.21	0.51
512	1.19	0.51	1.10
1024	1.67	0.99	1.87
2048	4.76	1.82	3.73
4096	9.39	3.67	8.98
8192	13.22	5.98	19.89

Table 1: This table shows the time measure it takes to perform an operation add, look up and remove to a map as a list.

table

table

n(us)	**add**	**look up**	**remove**
16	0.06	0.05	0.07
32	0.13	0.07	0.21
64	0.13	0.06	0.15
128	0.31	0.16	0.41
256	0.11	0.12	0.5
512	0.18	0.09	0.17
1024	0.23	0.19	0.59
2048	0.17	0.11	0.19
4096	0.67	0.13	0.58
8192	0.23	0.15	0.29

Table 2: This table shows the time measure it takes to perform an operation add, look up and remove to a map as a tree.

As we see from the table , depending on the structure they are created from, the tree structure executes more quickly than the list and that is because the complexity time of a list is O(n) to add an element or to remove an element or to look up an element, whereas that of a tree is O(n) to add elements and remove elements that is less than 32 elements, and O(log n) for more than 32 elements.

In the look up the list constantly needs to compare the new element with all the smaller items to determine the the right position so it takes time more than the tree while the tree always takes log(n) before adding a member.