Van Emde Boas Tree

Sadiya Ahmad

April 2020

1 Introduction

Van Emde Boas Tree , also know as vEB tree is a tree data structure which implements associative array with N-bit integer keys. It performs all the operations in O(log N) times. Van Emde boas tree has a depth of O(log log N) so they take exponentially less time to search through compared to binary search tree.This is because of its recursive nature.In short, if you choose to fall from a tree choose Van Emde Boas tree to fall from, it will hurt the least.

The idea of Van emde boas tree is just like indicating there exist a very large collection of integers entered into a predefined array. This array is superimposed with the tree storing the metadata or identification of existence of these elements in the array.

The metadata proves to be extremely helpful and most information can be obtained from it without even traversing through the actual array. For our experiment, in order to find out time spend in performing each query on hash map base Van emde boas tree we calculate the time at start and end of the function calling and record all the function call time into one variable.

There are enhancements made into the project with the hope of extracting more information from the metadata without examining the actual array.

2 Implementation

The technique applied on Van Emde boas tree is called cluster galaxy decomposition. To define these terms Galaxy is the part of the universe and exist in the power of two. this galaxy is further divided into clusters. Each layer of a Van Emde boas tree has following attributes:

Cluster: This has a size of sqrt(U) where each U is the size of the universe. The cluster C stores elements that represent the next level of the tree.

Summary: This stores clusters C with atleast one lement as an identifier of C.

Summary is a Van Emde boas node over a reduced universe sqrt(U).

Min/Max: The are the minimum and maximum values respectively of each layer stored in each layer next to summary.

Over this data structure we apply different functions. Functions that best work with Van Emde boas trees are:

1.Insert
2.Delete
3.Find
4.Minimum
5.Maximum
6.Predecessor
7.Successor

Universe U is defined and a galaxy is obtained in the power of 2 nearest value smaller than U. No duplicate element is entered in the tree. If an attempt is made to enter duplicate element we can just just ignore it.Theres are many options to implement Van emde boas tree . Some of them are priority queue , Hash tree e.t.c. For this project, Van emde boas tree is implemented using Hash map with map functions.The time is measured in nanoseconds. Van emde boas tree uses a unsigned bitwise operations to calculate the initial size of the cluster.

2.1 Insert

When inserting an element into the data structure two main cases occur: Assume —S— is the number of elements to be inserted in the data structure then if —S—¡2 there is no need for a recursive insert else if —S—¿=2 we proceed to recursive insert. For —S—¿2 we have further have two cases:
:

Case 1: Inserting a duplicate value.This can be ignored i.e. if a key already exist in hash map then return nothing,
Case 2: Inserting a new value.For this put() function of a map is called.

As we insert the values we check if they are minimum values or maximum values and replace min and max accordingly.If the number of elements are less than 2 then simply enter just one elemnent and assign it as min and max. Initial values of min and max are set to NIL=(-1).

Algorithm 1 Insert an element in O(log log —U—) time

1.insert (T , q U S, w):
2. —S— = 0
3. if T . min = :
4. T . min = T . max = q
5. return
6.
7. one$_e$$lement = T.min == T.max$
8.
9.$if q < T.min$:
10.$swapqandT.min$
11.
12.$if q > T.max$:
13.$swapqandT.max$
14.
15.$|S| = 1$
16.$if one_e lement$:
$return$
18.
19.$|S| = 2$
20.$if T.summary =$:
21.$T.summary = newnode$
22.
23.$|S|2$
24.$c, i = split(q, w/2)$
25.$cluster = T$
$.create_o r_u se_c luster(c)$
26
27$newlycreatedcluster$
28$if cluster.min =$:
29$T.summary.insert(c, w/2)$
30
31$cluster.insert(i, w/2)$

2.2 Delete

This calls for two possibilities, either completely deleting the tree or else remove one element of the tree.

Algorithm 2 Delete an element in O(log log —U—) time

```
1 delete (T , q  S, w):
2 c, i = split (q, w/2)
3
4  at most two elements
5 if T . summary = :
6 T . delete—S—2 (q, w)
7 return
8
9 T . delete—S—¿2 (q, w)
```

2.3 Find

This is rather a simplistic operation and the one that forms the bases of all others. As we iterate through the tree we can find the we call upon hash map predefined overridden function map.contains(key). This will help us to find the value that has been passed through another function.

In this experiment we call upon keyset() function and which runs all functions of the map and traverse through the tree.

Algorithm 3 Find an element

```
1.Find(key,value)
2.object.keyset();
3.   keyset() implements set¡Integer¿
4.   keyset().contains(obj o)
5.         map.containskey(o)
6.   End
7.End
```

2.4 Minimum/Maximum

Separated from the summary these are simply stored as metadata of each layer.For obtaining minimum to start at the root and move towards the leaves always taking the left most node containing 1. For maximum, we move from root to leaves taking right most node containing 1.

2.5 Predecessor

Finding a predecessor become easier in a Van Emde boas tree since it ignores the whole tree that does not contain any element. The formula is to start at the leaf node of which we are finding the predecessor and move up taking the **right branch** with 1 in it until we find a node that has 1 in its **left child**.

2.6 Successor

For successor, we are required to start at leaf node and move up to the left until we find a node that has a 1 in its **right child**.

3 Enhancement

Besides the primitive approach for Van Emde boas trees, quite a few enhancements can be made to the structure of the Van Emde boas trees.As described earlier we are storing min, max values along with summary at every level. With that said, we can also store sum of all elements , mean and median just beside the summary. This will help us in different calculations without traversing the Van Emde boas tree.

The concept of finding mean is achieved by storing the sum of of all the values while we input them and counting the unique values added to the data structure.

While finding mean was a fairly easy process , it seems calculating median

Algorithm 4 Find mean of all elements

```
Require: Variables sum=0, entries=0
  1.object.put(key,value)2.
  if map.contains(key) then
    ignore entry
  else
    entries++
    sum = sum+value
    map.put(key,value)
  end if
  3.mean = sum/entries
```

in O(log log n) time can pose a real challenge.Median is the middle element of the sorted array. We do not wish to traverse through the whole array which will cost us unreasonably O(u) time. So,instead of finding the median element we will focus on finding the position of median element. This will be a recursive process in which we will ignore the clusters that has fairly small size and holds no possibility of having a median in it. It is a complicated scenario since we

can get the value right away. In any case , this method shall take more time in execution than mean. The time for execution of mean is in nanoseconds.This meathod goes through a recurssive function within a for loop and finally end up calling getvalue function for map. The proposed theory to accomplish median is as follows:

Algorithm 5 Find median for each summary

```
Require: entries
  median-position=entries/2
  int Median(int median-position) 1.Find (summary-elements)
  2.Find (cluster-elements)
  3.
  for int i=0;i¡=summary-elements;i++ do
    if cluster[i].cluster-element¿0 then
      if cluster[i].cluster-element¡entries then
        median-position=subtract(median-position,cluster[i].cluster-element)
        Median(median-position)
      end if
    else
      return map.getvalue(median-position)
    end if
  end for
```

For instance, we have a universe size in power of 2 as 128, the number of elements entered are 17. Assuming that the array is sorted , the median value shall occur at 17/2 i.e approx 9th position. We will look into the first cluster of our first summary that has elements in it. If the elements in the cluster are lesser than 9 then 9th element can not exist in that cluster and therefore we will perform subtraction(9,cluster[i].cluster-elements). Suppose cluster-elements=3 then then in the next cluster ,the next position starting from the last position of the previous cluster we shall look for (9-3=6th) element. This will go on until the position(median-position) we are looking for is reduced to 1 or equal to cluster[i].cluster-element. Median-position at this moment will be the actual position of our median within that cluster.

4 Run time Analysis

As from the initial attempt of calculating mean and median, it appears calculation of mean takes significantly less time while execution of median is an exhaustive process. The time for execution of mean is distributed to the map.put function. Following are the initial results:

Time complexity of sum is O(1) and sum/entries is O(1), so overall complexity of mean is O(1). In case of median, time complexity of for statement is O(N) and time complexity of the if statement will be the linear function of

```
Universe Size is: 128
Seed = 1660420273385510
Universe Size is: 128
VanEmdeBoasHashMap.put in 753098 nanoseconds.
VanEmdeBoasHashMap mean in 5987 nanoseconds. Mean is 70
```

```
VanEmdeBoasHashMap median in 26586821 nanoseconds.
VanEmdeBoasHashMap iteration in 3952806 nanoseconds.
VanEmdeBoasHashMap.get in 86321530 nanoseconds.
VanEmdeBoasHashMap.remove in 49232810 nanoseconds.
VanEmdeBoasHashMap total time: 166853052 nanoseconds.
```

Figure 1: Experiment Results

the input size i.e O(1), so the total time complexity of the median function will be O(N). Since the median traverse through the whole tree , the complexity to traverse through the tree is O(log log N).The median time shown in the result above is not from the implementation of complete median function.

The time complexity of computing minimum and maximum is always O(log N) while insertion, deletion, find take O(log log N) time.

5 Application and Advantages

It is well proven that vEB tree are one of the fastest tree structures in the market , however they are not the most frequently used ones. This makes us think why is that so ? To answer this question we must understand overheads of constructing a tree. Major characteristic of a tree is formation of its branches. In case of a binary tree at each level we are creating two nodes, while in van Emde boas nodes are not defined.Initially each element added creates a tree in Van Emde boas structure.This takes a enormous amount of space but is equally beneficial when dealing with humongous data. Generally in applications we do not deal with that amount of data and hence the over head of using Van Emde boas tree with small data is huge.

Theoretically, Van Emde boas tree can be used in location apps where among the group of restaurants/offices e.t.c we need to find location and distribute the work.Van Emde boas tree can also be a good option in high performance routers, particularly solving packet classification problem.As explained in [4] the idea is based on a pre-computed decision tree which is traversed for each packet that needs to be classified.

Once space complexity of Van Emde boas trees is reduced, this algorithm shall demand for further evaluation and perhaps improved version of the algorithm can be attained.One of the requirements might include to statistically compare the rate of search in the tree. If the statistical data like mean , median , mode and standard deviation is presorted in the data structure , the statistical evaluation will become far more easier. The experimental enhancement performed in this project is a prototype to measure the time taken to store and modify such meta data into the data structure. Through this number of graphs can be obtained.

6 Conclusion

The preliminary algorithms have been successfully implemented in the Van Emde boas tree using Hash Map and there time of execution is calculated.The mean of the data is calculated but the calculation of mean is still under process. This is due to the lack of understanding of excessive recursive nature of the algorithm. Median implementation is a time consuming process but is not unachievable.

References

[1] van emde boas data structure. 2012.

[2] Prof. Erik Demaine. Advanced data structures. 2012.

[3] Marcel Ehrhardt. An in-depth analysis of data structures derived from van-emde-boas-trees. 2015.

[4] Grigore Rosu Sumeet Singh Florin Baboescu, Dean M. Tullsen. A tree based router search engine architecture with single port memories. 2005.

[5] Gerth Stølting Brodal Rolf Fagerberg Riko Jacob. Cache oblivious search trees via binary trees of small height. 2001.

[6] Hao Wang Bill Lin. Pipelined van emde boas tree: Algorithms, analysis, and applications. 2007.

[7] Eduardo Laber David Sotelo. van emde boas tree. 2007.

[8] Mary K. Vernon. Complexity and big-o notation.