Van Emde Boas Tree

Shubham Agrawal

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Agenda:

1. Node Structure
2. Insertion
3. Successor
4. Delete
5. Time Complexity
6. Space Complexity

Problem

Given a set **S** of elements such that the elements are taken from universe {0, 1, …. u-1}, perform following operations efficiently.

1. Insert
2. Successor
3. Delete

Different Solutions:

1. Bit Vector

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

2. Balanced BST like AVL tree, Red Black tree
3. VEB tree

Complexity Comparison:

Here, **u**-->universe size, **n**--> Number of elements present in the tree.

Data Structure	Insert	Successor	Delete
Bit Vector	O(1)	O(u)	O(1)
AVL (BST)	O(log n)	O(log n)	O(log n)
VEB	O(log log u)	O(log log u)	O(log log u)

Where might O(log log u) come from

T(k) = T(k/2) + O(1) ------------> O(log k)

T(log u) = T (log(u) / 2) + O(1) ---------> O(log log u)

S(u) = S(sqrt(u)) + O(1)   ---------------> O(log log u)

1. Bit Vector

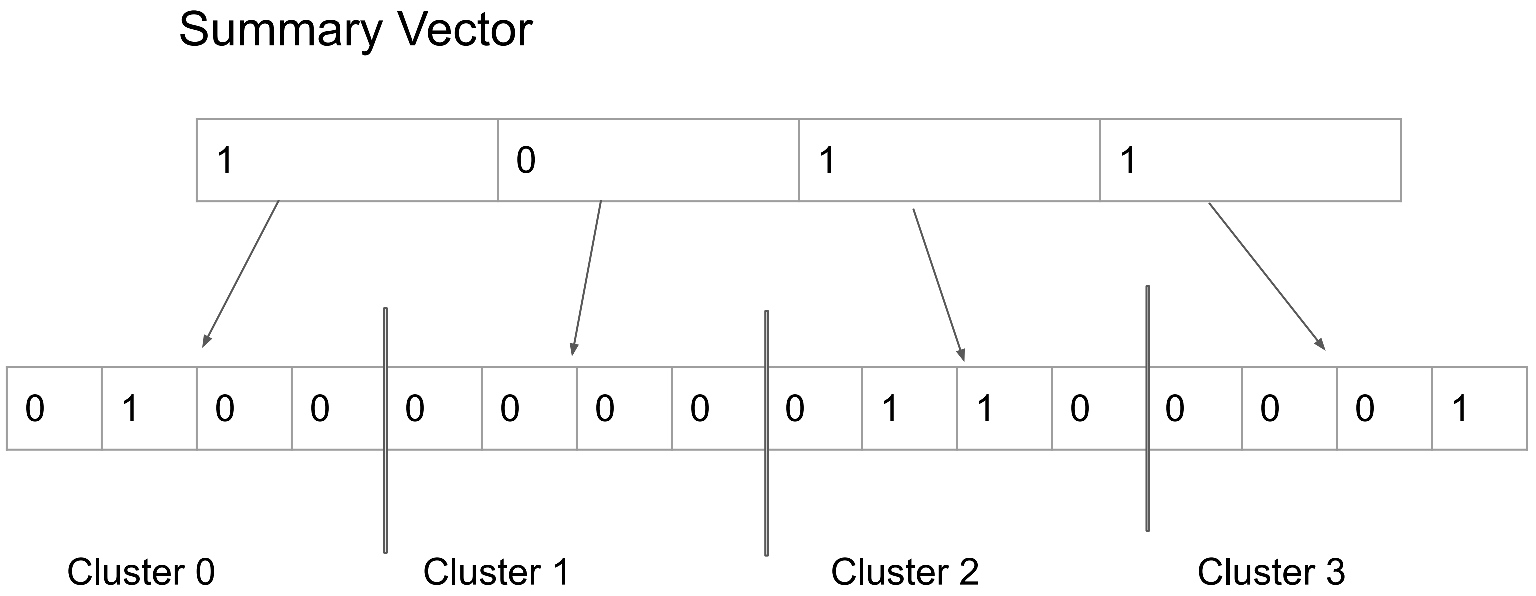
0 -> Absent, 1 -> Present

Suppose u = 16 elements from {0, 1, …, 15}

And n = 4 and {1, 9, 10, 15}

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

2. Split the universe into sqrt(u) clusters of size sqrt(u)



1. Insert O(1)
2. Successor(X):
    a. Look in X’s Cluster ---> O(sqrt(u))
    b. Look for the next 1 in summary vector ---> O(sqrt(u))
    c. Look for the first one in that cluster ---> O(sqrt(u))

O(sqrt(u))

Terminologies

If $x = i * \mathrm{sqrt}(u) + j$, where $0 <= j < \mathrm{sqrt}(u)$

i-> cluster number

j-> position in that cluster

$X = 9$ $\mathrm{sqrt}(16)=4$, $9 = 2*4+1$

9 ----> 10 01

1. **high(X) = floor(X/sqrt(u)) = 2**
2. **low(X) = X % sqrt(u) = 1**
3. **index(i, j) = i*sqrt(u) + j ---> index(2, 1) = 2*sqrt(16)+1 = 2*4+1=9**

3. Recursion

We define data structure V (size u) VEB.

It contains:

1. V.clusters[i] from 0 <= i < sqrt(u) size(V.cluster[i]) = sqrt(u) VEB
2. V.summary size(V.summary) = sqrt(u) VEB

Structure of vEB node

10 ----> 1, 3, 43, 34, 23 , 100 ---> max = 100 → 2^8

```
Class VEB{

    int u;

    vector<VEB*> clusters(sqrt(u));

    VEB* summary(Sqrt(u));

};
```

Insertion Algo:

Insert(V, x):

Insert(V.cluster[high(x)], low(x))

Insert(V.summary, high(X))

$T(u) = 2*T(\sqrt{u}) + O(1)$

O(log u)

Successor Algo:

```
Successor(V, X):
    i = high(X)
    j = Successor(V.cluster[i], low(X))
    If j == Infinity:
        i = Successor(V.summary, i)
        j = Successor(V.cluster[i], -Infinity)   // V.cluster[i].min
Return index(i, j)
```

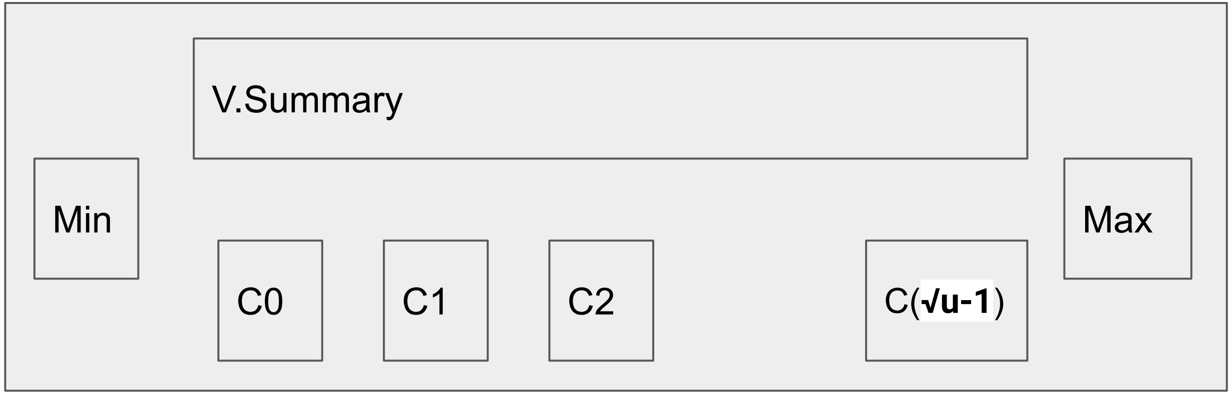
$T(u) = 3*T(\sqrt{u}) + O(1)$

$O(\log u)^{(\log 3)}$

4. Store Min and Max

Augmentation like AVL tree :

Node Structure of VEB Tree



Modified Insertion Algo:

Insert(V, x):

if(X< V.min):

V.min = X

if(X> V.max):

V.max = X

Insert(V.cluster[high(i)], low(x))

Insert(V.summary, high(X))

O(log u)

Modified Successor Algo:

```
Successor(V, X):
    i = high(X)
    if( low(X) < V.cluster[i].max):
        j = Successor(V.cluster[i], low(X))
    else:
        i = Successor(V.summary, i)
        j = V.cluster[i].min
    Return index(i, j)
```

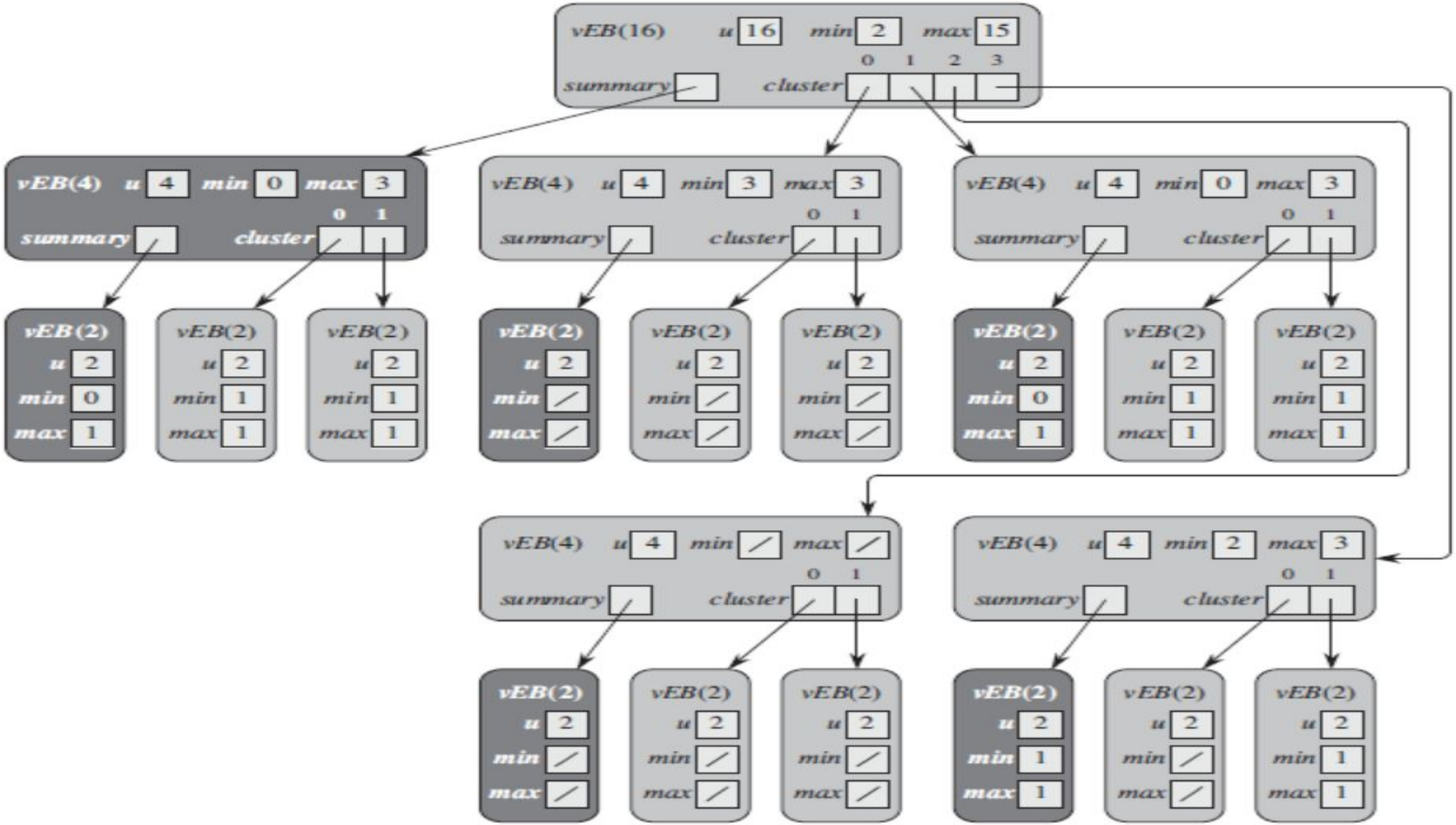
$$T(u) = T(\sqrt{u}) + O(1)$$

$$O(\log \log u)$$

5. Don’t Store min recursively

```
Insert(V, x):
    If V.min = None:
        V.min = V.max = X
        return
    if(X< V.min):
        Swap V.min  <->  X
    if(X> V.max):
        V.max = X
    If V.cluster[high(X)].min = None:
        Insert(V.summary, high(X))
    Insert(V.cluster[high(i)], low(x))
```

O(log log u)



vEB(16) u 16 min 2 max 15
summary
cluster
0 1 2 3
vEB(4) u 4 min 0 max 3
summary cluster
0 1
vEB(4) u 4 min 3 max 3
summary cluster
0 1
vEB(4) u 4 min 0 max 3
summary cluster
0 1
vEB(2) u 2 min 0 max 1
vEB(2) u 2 min 1 max 1
vEB(2) u 2 min 1 max 1
vEB(2) u 2 min max
vEB(2) u 2 min max
vEB(2) u 2 min max
vEB(2) u 2 min 0 max 1
vEB(2) u 2 min 1 max 1
vEB(2) u 2 min 1 max 1
vEB(4) u 4 min max
summary cluster
0 1
vEB(4) u 4 min 2 max 3
summary cluster
0 1
vEB(2) u 2 min max
vEB(2) u 2 min max
vEB(2) u 2 min max
vEB(2) u 2 min 1 max 1
vEB(2) u 2 min max
vEB(2) u 2 min 1 max 1

Delete Algo:

```
Delete (V, X):
    If X=V.min:
        I = V.summary.min
        If i=None:
            V.min=V.max=None
            Return
        X = V.min = index(i, V.cluster[i].min)
    Delete(V.cluster[high(X), low(X))
    If V.cluster[high(X)].min = None:
        Delete(v.summary, high(X))
    If X = V.max:
        If V.summary.max = None:
            V.max = V.min
        Else:
            i = V.summary.max
            V.max = index(i, V.cluster[i].max)
```

O(log log u)

Space Complexity : **O(u)**

References:

1. https://www.geeksforgeeks.org/van-emde-boas-tree-set-1-basics-and-construction/
2. https://www.youtube.com/watch?v=hmReJCupbNU
3. https://www.youtube.com/watch?v=xMr_EhBqBpw
4. Introduction to Algorithms - Third Edition (CLRS book) page 531.