RecdV 3-1.

25/25

25/25

20.3-1

Here, my solution would be to store an integer at each node where a key is stored and for each single key this is initially 1 and can be incremented. For adding a key to vEB we take normal procedure as insert but when we find the right place if there is already key increment it, else add it. For delete if keys counter is not 1 then decrease by 1 else delete the key. Also increase other counters associated with vEB if not neressary. Successor operation is same, doesn't require change.

3/25

20.3-2

Each node also stores two floats x and y coordinates. Placement is determined with last digit of x is high(key) and last digit of y is low(key) this criteria determines where to place the key in VEB. Looking at last digit is to decrease the possible number of same value keys.

3-1. a)

```
vEB-Tree-Member
    if x==V.min or x==V.max
        return true;                    // no change necessary
    elseif V.u==2
        return false
    else return vEB-Tree-Member(V.cluster[high(x)], low(x))
```

```
vEB-Tree-Successor(V, x)
    if V.u==2
        if x==0 and V.max==1
            return 1
        else return NIL
    elseif V.min ≠ NIL and x< V.min
        return V.min
    else max-low = vEB-Tree-Maximum(V.cluster[high(x)])
        if max-low ≠ NIL and low(x) < max-low
            offset = vEB-Tree-Successor(V.cluster[high(x)], low(x))
            return index(high(x), offset)
        else succ-cluster = vEB-Tree-Successor(V.summary, high(x))
            if succ-cluster == NIL
                return NIL
            else offset = vEB-Tree-Minimum(V.cluster[succ-cluster])
                return index(succ-cluster, offset)
```

After checking the pseudocode, seems like no change is necessary since the basic principle is same and logic with operations such as insert, successor etc. are again same. But complexity should change.

$n = \log v$

$T(n) = 2T(n^{\frac{1}{3}}) + c$ $\quad$ $T(\log v) = 2T(\frac{1}{3}\log v) + c$

- x
- $\frac{2}{3}x$
- $\frac{4}{9}x$

$x + x\frac{2}{3}(1 + \frac{2}{3} + (\frac{2}{3})^2 \cdots)$

$\frac{1}{1-\frac{2}{3}} = 3$

$\log_{\frac{3}{2}} n$

n
$n^{\frac{2}{3}}$
$n^{\frac{4}{9}}$
$n^{\frac{8}{27}}$
$\vdots$
1

count = $\log_{\frac{3}{2}} n$ $\quad$ sum = count $(n + n^{\frac{2}{3}} + n^{\frac{4}{9}} + n^{\frac{8}{27}} \cdots 1)$

2/7

0/18

b) For successor, I would store max in 2D array such that in arr[i][j] is store max of the biggest clusters and j stores max values in each cluster that is in ith cluster.

```
successor(v, x)   // from our lecture
    i = high(x)
    if low(x) < arr[i][j]   // j current for our cluster
        j = successor(v.cluster[i], low(x))
    else
        ...
    return index(i, j)
```