Homework Assignment-6

Dhayalini Nagaraj
A20359686

1. Given, there integer d, $1 \leq d \leq n$. We need to determine d with as few drops as possible given some m no. of iphones.

Van Emde Boas tree uses divide and conquer method.

Since we have n floor and m phones, we can drop from $n/2$ floor and if it breaks we can drop from floors which are less than $n/2$. If it doesnot break then we can drop from floors higher then $n/2$ floors.

Since we have only m phones only m phones ~~times~~ we can drop ~~a phone~~. and cannot be reused if it breaks.

~~This~~ Inorder to find d with few drops.

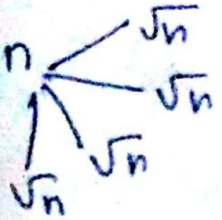
Let us divide the ~~data~~ range $\{0, \ldots n-1\}$ into blocks of size $\sqrt{n}$.

This $\sqrt{n}$ is known as clusters.

There is ~~a~~ a summary structure which keeps tracks of clusters that are non empty.

we initially set v. summary as NIL

First let us drop phone from $\sqrt{n}$ floor



So time taken will be $O(\lg \lg n)$

Let us assume $n = 2^{2k}$ for some natural k.

we can super impose a tree of degree $n^{1/2}$

* Drop phone from $\sqrt{n}$ floor
* if it breaks then drop phone from $n^{1/4}, n^{1/8} \ldots 2$
  The vEB tree depth is down to size 2
* if it does not break the drop phone from $2*\sqrt{n}$, $3*\sqrt{n}$, $4*\sqrt{n}$ .... $\sqrt{n}*\sqrt{n}$ (clusters.maxchild-floor)

<u>Algorithm</u>: vEB-PHONEDROP(m,n)

// m - no.of phone, n - no. of floors

clusters = floor(sqrt(n))

set i = 1, j = 1, d = NIL

while i ≤ clusters do
if m ≠ 0 then // If m = 0 then the loop ends and return d
if drop (phone, V.clusters[i].max)

// Phones denotes a constant variable, V.clusters[i].max
// denotes the max child of the cluster.
// The function drop return true or false if the
// phone breaks or not.

return true

d = V.clusters[i].max

~~if m ≠ 0 then // If m = 0 then d is returned.~~

set m = m-1.

set n = clusters

if n ≠ 2 // not equal to min depth of tree

Clusters = floor(sqrt(n))

~~[illegible]~~

set i = 1

else

child = V.cluster[i].child

// returns no. of child elements in the
// cluster

~~for [illegible]~~

while j ≤ child do

if drop (phone, V.cluster[i].child[j])

return true

d = min(V.cluster[i].child[j], d)

else

j++

end if

end while

~~end if~~

```
        endif
    else
        i++
    end if
    endif
end while.
return d.
```

The value returned from algorithm gives the value of d, That is phone dropped from d or higher floor will be destroyed.

This algorithm is executed until $m \neq 0$ and and with few drops finding d using recursion.

If m is not mentioned, then it can be considered until we find the exact floor from ~~the~~ where when iphone is dropped, it get destroyed.

Since m is mentioned, we have set a constraint that m should not be zero and found d, such that $1 \leq d \leq n$ and iphone gets destroyed when dropped from d floor and higher floors.

2). Inserting an element

PROTO-VEB-INSERT : It usually makes two recursive calls

* one to insert the element and
* to insert the element's cluster number into the summary.

VEB-TREE-INSERT : This procedure will make only one recursive call.

VEB-EMPTY TREEINSERT : Do not need any recursive calls to insert ~~into~~ element into an empty VEB TREE

For the new requirement:

VEB-EMPTY TREEINSERT (V, x) :

The procedure for inserting an element into an empty VEB TREE does not change. because the code ~~that~~ is dealt with the case V.max explicilly and it is correct for the new ~~required~~ requirement also.

VEB-EMPTY-TREE-INSERT (V, x)

V.min = x

V.max = x

VEB - TREE - INSERT: Let us assume that x is not already an element in VEB-TREE. We should V.max in a similar way to V.min for the new requirement.

Algorithm: vEB-TREE-INSERT (V, x)

```
if V.min == NIL then
    vEB-EMPTY-TREE-INSERT(V, x)
else
    if x < V.min then
        exchange x with V.min
    end if
    if x > V.max then
        exchange x with V.max
    end if
    If V.u > 2
        if vEB-TREE-MINIMUM (V.cluster[high(x)]) == NIL
            VEB-TREE-INSERT (V.summary, high(x))
            VEB-EMPTY-TREE-INSERT (V.cluster[high(x)],
                                   low(x))
        else VEB-TREE-INSERT(V.cluster[high(x)], low(x))
        endif
    endif
endif
```

Explaination:

V.min and V.max are handled in the begening and then the insertion is done.

Deleting an element

Let us assume that x is currently an element in the set represented by the vEB tree V. In this V.min and V. max are handled in similar way and then the deletion occurs after that.

~~vEB~~ Algorithm: vEB - TREE - DELETE (V, x)

```
if V.min == V.max
    V.min = NIL
    V.max = NIL
else if V.u == 2
    if x == 0
        V.min = 1
    else V.min = 0
    V.max = V.min
    end if
end if
else if x == V.min
    first-cluster = VEB-TREE-MINIMUM (V.summary)
    x = index (first-cluster,
        VEB-TREE-MINIMUN (V.cluster [first-cluster]))
    V.min = x
end if
else if x == V.max then
    last-cluster = VEB-TREE-MAXIMUM (V.summary)
```

Scanned by CamScanner

x = index (last-cluster , VEB-TREE-MAXIMUM (V.cluster [last-cluster]))

V.max = x

endif

VEB-TREE-DELETE (V.cluster [high(x)] , low(x))

if VEB-TREE-MINIMUM (V.cluster [high(x)]) == NIL

VEB-TREE-DELETE (V.summary , high(x))

end if

endif

3) The number n of elements are stored in tree rather than on the universe size u. Assume $\sqrt{u}$ is always an integer.

a) There are $\sqrt{u}$ clusters and we have 1 summary. Each of it require $P(\sqrt{u})$ space. We should store the size, the array of pointers, the min and the max. To store all this we need $\Theta(\sqrt{u})$ space. So

$$P(u) = (\sqrt{u}+1)\,P(\sqrt{u}) + \Theta(\sqrt{u})$$

b). To prove than recurrence $P(u) = (\sqrt{u}+1)P(\sqrt{u}) + \Theta(\sqrt{u})$ has the solution $P(u) = O(u)$

Let us assume

$P(u) = a(u-2)$, where a is some constant

then

$$P(\sqrt{u}) = a(\sqrt{u}-2)$$

Now we have

$$P(u) = (\sqrt{u}+1) \cdot a(\sqrt{u}-2) + \Theta(\sqrt{u})$$

$$= a(u-2) - a\sqrt{u} + \Theta(\sqrt{u})$$

$$\leq a(u-2)$$

We should choose 'a' such that it is larger than the constant factor hidden in $\Theta(\sqrt{u})$. So we can say

$P(u) = a(u-2)$ is correct.

~~Now~~ Then, $O(a(u-2)) = O(u)$

So we get

$$P(u) = O(u).$$

Hence Proved.

i) Given: All of the array-of pointers substructures can be stored in a single array outside the vEB tree itself.

Let us assume $P(u) = a(u-2)$ and $P(\sqrt{u}) = a(\sqrt{u}-2)$

With the given assumptions we have

$$P(u) = (\sqrt{u}+1) \cdot a(\sqrt{u}-2) + O(1)$$

$$\cancel{= a(u-2) - a\sqrt{u} +}$$

$$= a(u-2) - a\sqrt{u} + O(1)$$

$$\leq a(u-2)$$

This is because when u is large we can find a 'a' such that $a\sqrt{u}$ is larger than the constant $O(1)$.

So moving the array-of pointers substructure outside vEB tree will not improve the the vEB structure.

It will not improve the vEB structure.

Scanned by CamScanner