VAN EMDE BOAS TREE

1) Enfoques preliminares

- Direct addressing
- Superimposing a binary tree structure
- Superimposing a tree of constant height

❖ Direct addressing

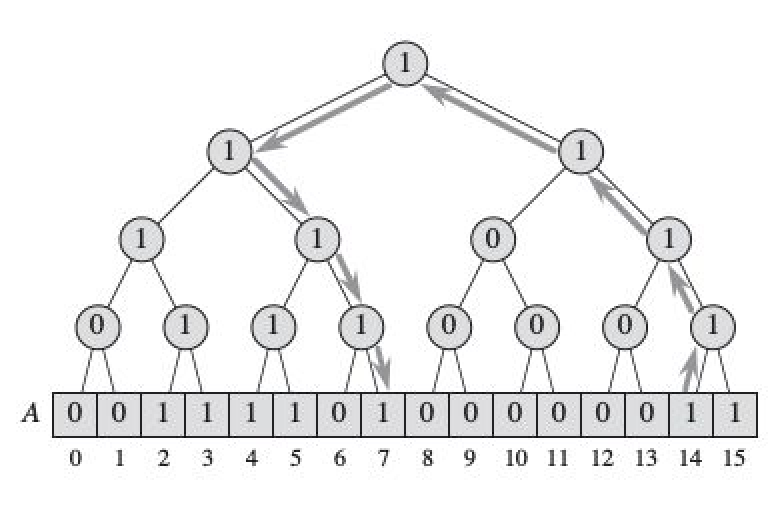
Para almacenar un conjunto dinámico en un vector de bits:

U={0,1,2,...,u-1}. A=[0,...,u-1], A[x] =1 si x pertenece al conjunto.

La inserción y la eliminación se dan en O(1).

Mínimo, máximo, sucesor y predecesor son θ(u) en el peor caso.

❖ Superimposing a binary tree structure



Cada nodo interno contiene un 1 si y sólo si alguna hoja en su subárbol contiene un 1(OR).

Mínimo, máximo, sucesor y predecesor, inserción y eliminación, son O(lg u) en el peor caso(altura del árbol=lg u).

Para un "u" muy pequeño respecto al universo, red-black es más rápido (O(lg n)).

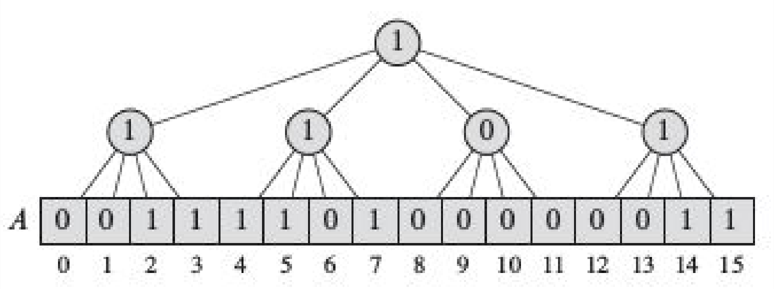
❖ Superimposing a tree of constant height

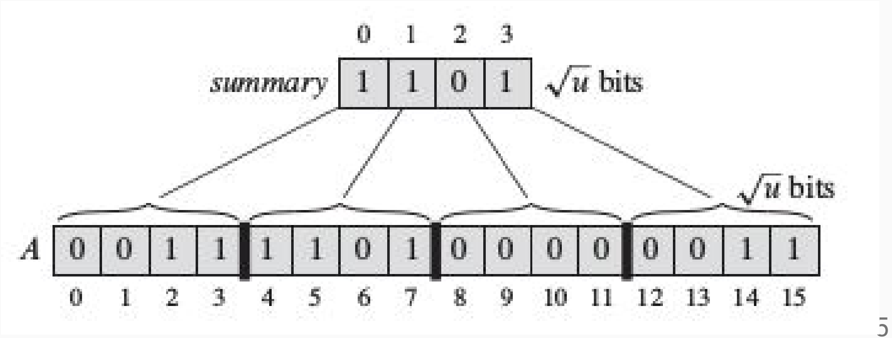
$U=2^{2k}$.

La altura del árbol siempre será 2.

Insertar: O(1).

Mínimo, máximo, predecesor, sucesor eliminar O(√u).





2) Una Estructura Recursiva

Se modifica la idea de tener un árbol de grado $\sqrt{u}$

Ahora se toman estructuras que mantienen $\sqrt{u} = u^{1/2}$ elementos, que a su vez sostienen estructuras de $u^{1/4}$ elementos las cuales sostienen estructuras de $u^{1/8}$ elementos y así sucesivamente.

$$T(u) = T(\sqrt{u}) + O(1)\,. \qquad (20.2)$$

$$T(u) = T(\sqrt{u}) + O(1)\ . \tag{20.2}$$

$$\begin{aligned} \text{high}(x) &= \left\lfloor x/\sqrt{u} \right\rfloor\ , \\ \text{low}(x) &= x \bmod \sqrt{u}\ , \\ \text{index}(x, y) &= x\sqrt{u} + y\ . \end{aligned}$$

2.1) Estructura prototipo Van Emde Boas

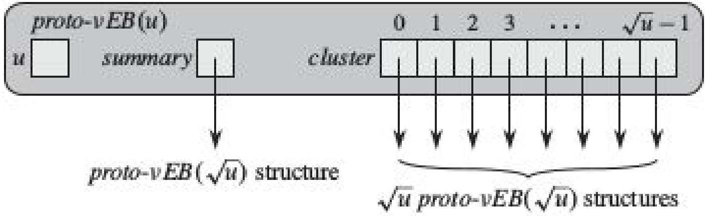
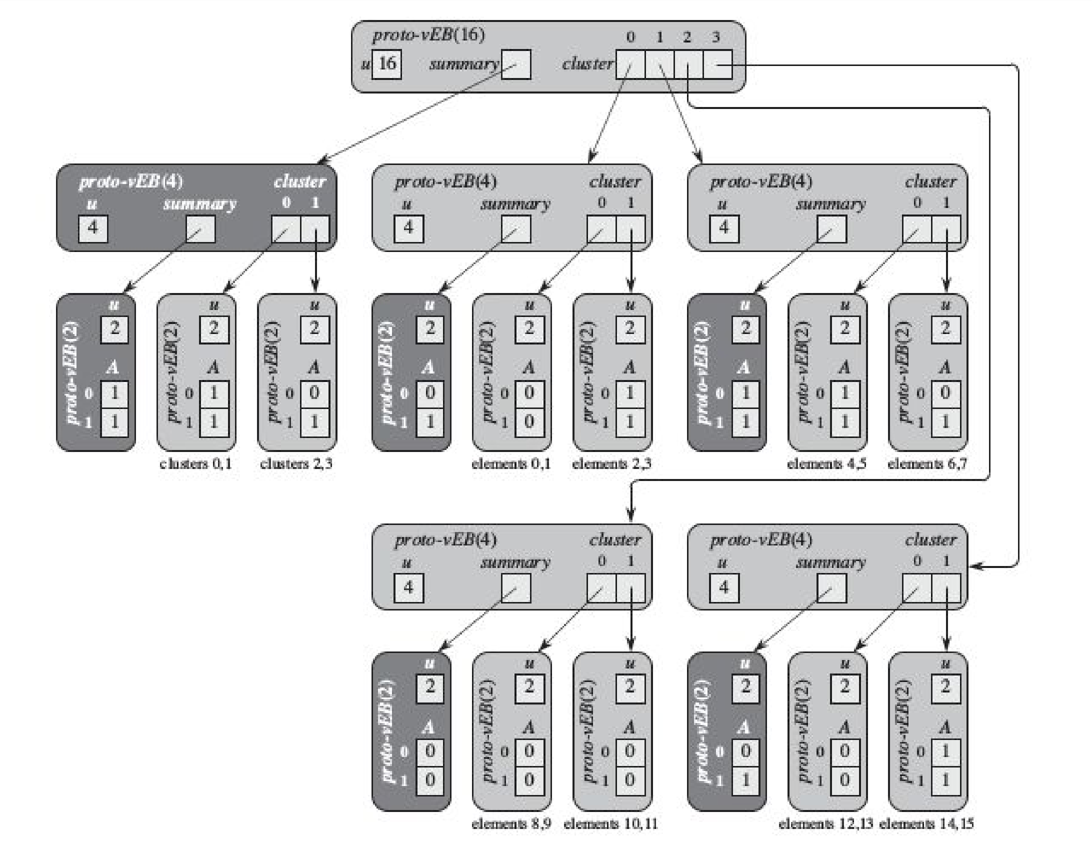


Figure 20.3 The information in a *proto-vEB*(u) structure when $u \geq 4$. The structure contains the universe size u, a pointer *summary* to a *proto-vEB*$(\sqrt{u})$ structure, and an array *cluster*$[0 \,..\, \sqrt{u}-1]$ of $\sqrt{u}$ pointers to *proto-vEB*$(\sqrt{u})$ structures.

Estructura



Estructuras que representa el conjunto de:

{2; 3; 4; 5; 7; 14; 15}

2.1.1) Determinar si un valor está en la estructura

```
PROTO-vEB-MEMBER(V, x)
if V.u == 2
    return V.A[x]
else return PROTO-vEB-MEMBER(V.cluster[high(x)], low(x))
```

2.1.2) Encontrar el mínimo elemento

```
PROTO-vEB-MINIMUM(V)
if V.u == 2
    if V.A[0] == 1
        return 0
    elseif V.A[1] == 1
        return 1
    else return NIL
else min-cluster = PROTO-vEB-MINIMUM(V.summary)
    if min-cluster == NIL
        return NIL
    else offset = PROTO-vEB-MINIMUM(V.cluster[min-cluster])
        return index(min-cluster, offset)
```

2.1.3) Encontrar el sucesor

```
PROTO-vEB-SUCCESSOR(V, x)
if V.u == 2
    if x == 0 and V.A[1] == 1
        return 1
    else return NIL
else offset = PROTO-vEB-SUCCESSOR(V.cluster[high(x)], low(x))
    if offset ≠ NIL
        return index(high(x), offset)
    else succ-cluster = PROTO-vEB-SUCCESSOR(V.summary, high(x))
        if succ-cluster == NIL
            return NIL
        else offset = PROTO-vEB-MINIMUM(V.cluster[succ-cluster])
            return index(succ-cluster, offset)
```

2.1.4) Insertar un Elemento

```
PROTO-vEB-INSERT(V, x)
if V.u == 2
    V.A[x] = 1
else PROTO-vEB-INSERT(V.cluster[high(x)], low(x))
    PROTO-vEB-INSERT(V.summary, high(x))
```

3) Los árboles Van Emde Boas

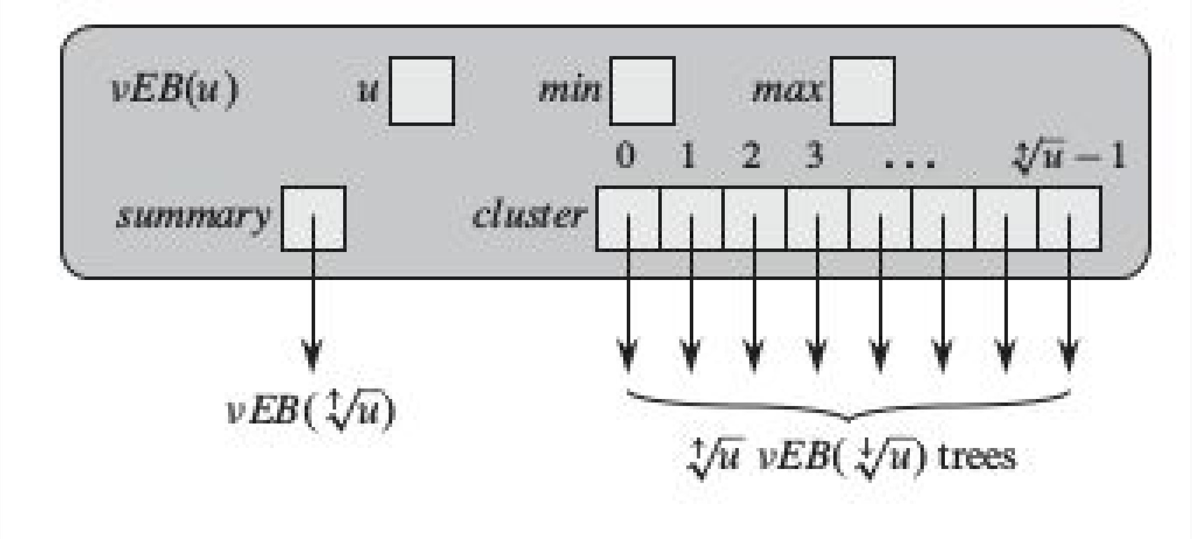
$U=2^k$

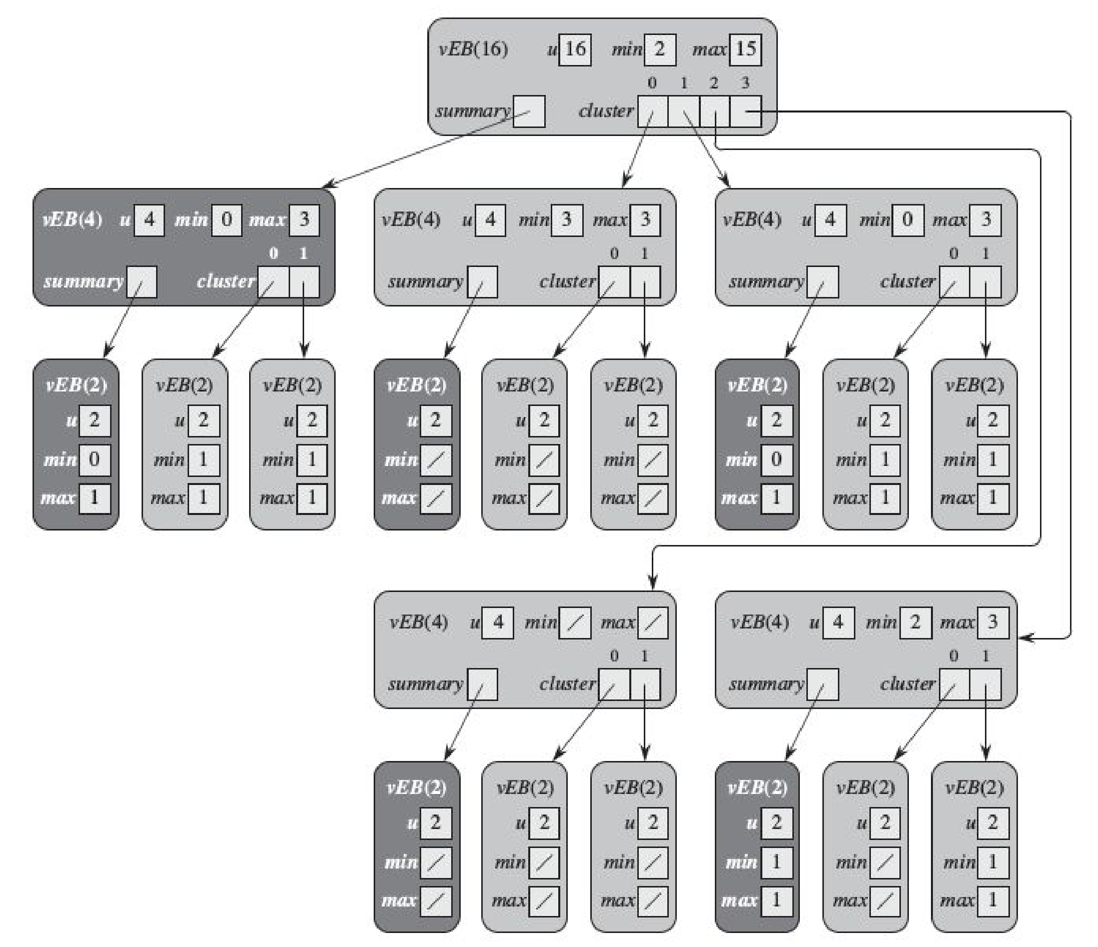
$$\lceil (\lg u)/2 \rceil \quad \lfloor (\lg u)/2 \rfloor$$

$$u = \sqrt[\uparrow]{u} \cdot \sqrt[\downarrow]{u}$$

$$\begin{aligned} \text{high}(x) &= \lfloor x / \sqrt[\downarrow]{u} \rfloor , \\ \text{low}(x) &= x \bmod \sqrt[\downarrow]{u} , \\ \text{index}(x, y) &= x \sqrt[\downarrow]{u} + y . \end{aligned}$$

3.1) Árboles Van Emde Boas





Estructura que representa el conjunto de:

{2; 3; 4; 5; 7; 14; 15}

$$m = \lg u,$$

$$T(u) \le T(\sqrt[4]{u}) + O(1)\,.$$

$$\lceil m/2 \rceil \le 2m/3$$

$$T(2^m) \le T(2^{\lceil m/2 \rceil}) + O(1)$$

$$S(m) = T(2^m),$$

$$T(2^m) \le T(2^{2m/3}) + O(1)$$

$$S(m) \le S(2m/3) + O(1)$$

$$\log_{3/2} 1 = \log_2 1 = 0.$$

$$T(u) = T(2^m) = S(m)$$

$$O(\lg m) = O(\lg \lg u).$$

3.2) Operaciones en un árbol Van Emde Boas

3.2.1) Maximo y mínimo

```
vEB-TREE-MINIMUM(V)
    return V.min

vEB-TREE-MAXIMUM(V)
    return V.max
```

3.2.2) Determinar si un valor está en la estructura

```
vEB-TREE-MEMBER(V, x)
if x == V.min or x == V.max
    return TRUE
elseif V.u == 2
    return FALSE
else return vEB-TREE-MEMBER(V.cluster[high(x)], low(x))
```

3.2.3) Encontrar el sucesor y predecesor

```
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)
```

```
vEB-TREE-PREDECESSOR(V, x)
if V.u == 2
    if x == 1 and V.min == 0
        return 0
    else return NIL
elseif V.max ≠ NIL and x > V.max
    return V.max
else min-low = vEB-TREE-MINIMUM(V.cluster[high(x)])
    if min-low ≠ NIL and low(x) > min-low
        offset = vEB-TREE-PREDECESSOR(V.cluster[high(x)], low(x))
        return index(high(x), offset)
    else pred-cluster = vEB-TREE-PREDECESSOR(V.summary, high(x))
        if pred-cluster == NIL
            if V.min ≠ NIL and x > V.min
                return V.min
            else return NIL
        else offset = vEB-TREE-MAXIMUM(V.cluster[pred-cluster])
            return index(pred-cluster, offset)
```

3.2.4) Insertar un Elemento

```
vEB-EMPTY-TREE-INSERT(V, x)
V.min = x
V.max = x
```

```
vEB-TREE-INSERT(V, x)
if V.min == NIL
    vEB-EMPTY-TREE-INSERT(V, x)
else if x < V.min
        exchange x with V.min
    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))
    if x > V.max
        V.max = x
```

3.2.5) Eliminar un Elemento

```
vEB-TREE-DELETE(V, x)
if V.min == V.max
    V.min = NIL
    V.max = NIL
elseif V.u == 2
    if x == 0
        V.min = 1
    else V.min = 0
    V.max = V.min
else if x == V.min
        first-cluster = vEB-TREE-MINIMUM(V.summary)
        x = index(first-cluster,
            vEB-TREE-MINIMUM(V.cluster[first-cluster]))
        V.min = x
    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))
        if x == V.max
            summary-max = vEB-TREE-MAXIMUM(V.summary)
            if summary-max == NIL
                V.max = V.min
            else V.max = index(summary-max,
                    vEB-TREE-MAXIMUM(V.cluster[summary-max]))
    elseif x == V.max
        V.max = index(high(x),
            vEB-TREE-MAXIMUM(V.cluster[high(x)]))
```