Introduction to Algorithms

Chapter 20: van Emde Boas 木 つづき

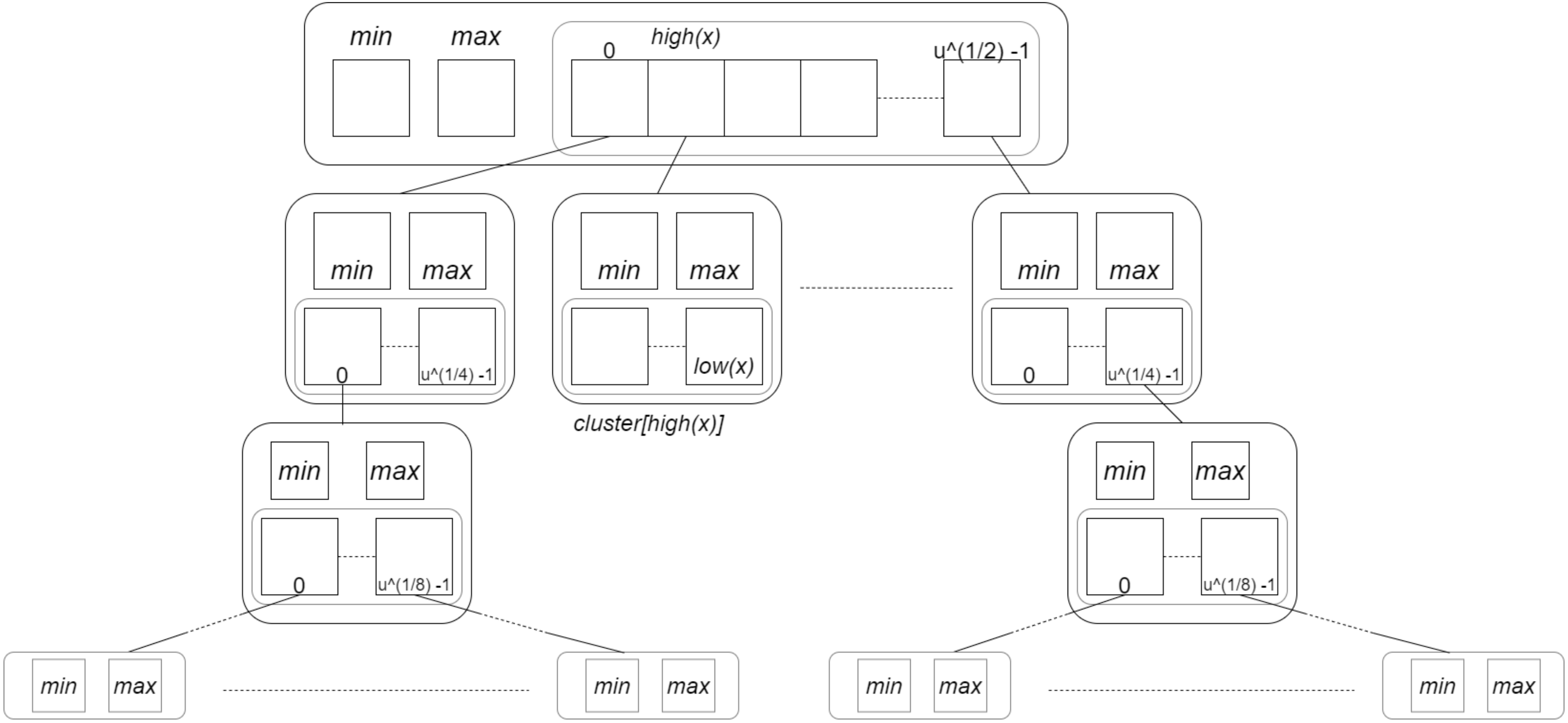
@ohken

2020/03/13

前回

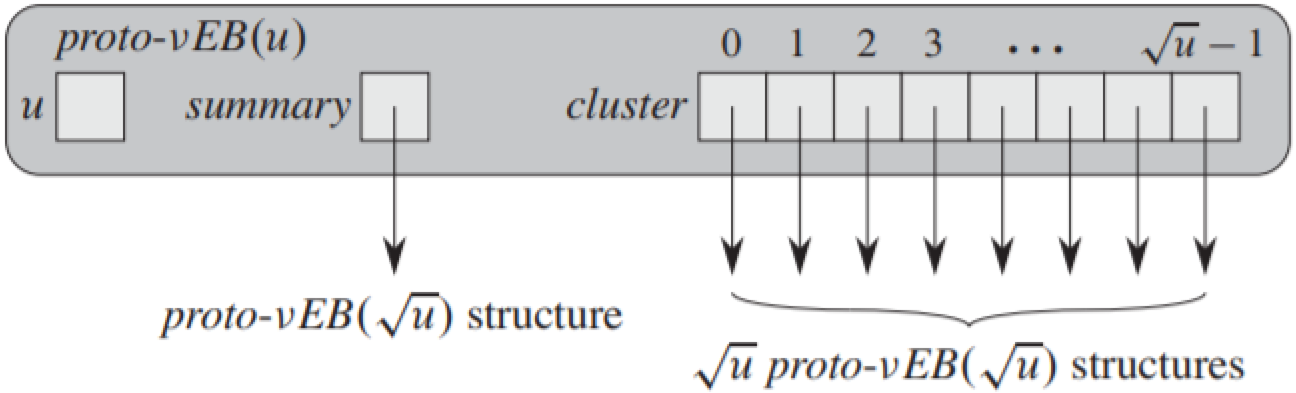
- この章の目的
  - $\text{Member, Successor}/\text{Predecessor, Minimum}/\text{Maximum, Insert, Delete}$ を集合に対して高速に行えるデータ構造を作る！
  - vEB木 : $\{0, \cdots, u-1\}$の部分集合のみを扱うとき$O(\log \log u)$で行える
- プロトタイプとして高さ$O(\log \log u)$の$proto\text{-}vEB$木を構成.

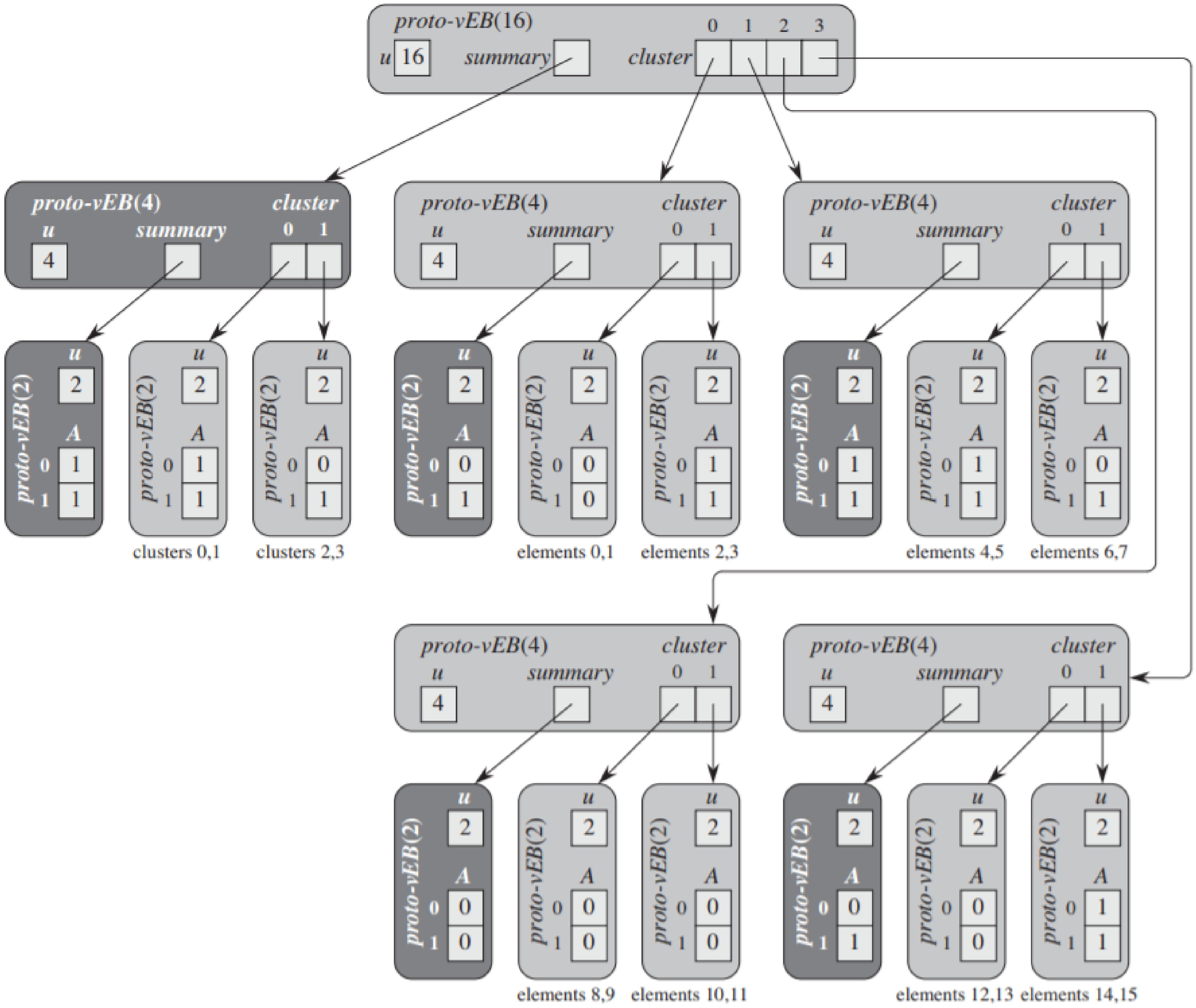
van Emde Boas 木



proto-vEBの定義

- $proto\text{-}vEB(2) = A[0..1]$: 長さ2の$\{0,1\}$-値配列
- $u = 2^{2^k}, k \geq 1$に対して$proto\text{-}vEB(u)$は次からなる:
  - $summary : proto\text{-}vEB(\sqrt{u})$のポインタ
    (木のノードに対応)
  - $cluster[0..\sqrt{u}-1]$: 長さ$\sqrt{u}$の$proto\text{-}vEB(\sqrt{u})$のポインタの配列
    (各ポインタが子ノードに対応)





proto-vEB(16)
u 16
summary
cluster
0 1 2 3
proto-vEB(4)
u
summary
cluster
4
0 1
proto-vEB(2)
u
2
A
clusters 0,1
clusters 2,3
elements 0,1
elements 2,3
elements 4,5
elements 6,7
elements 8,9
elements 10,11
elements 12,13
elements 14,15

記号の準備(再掲)

- $u = 2^{2^k}$, $0 \le x < u$
  - $\mathrm{high}(x) = \lfloor x/\sqrt{u} \rfloor$: xの上半分のビット(xを含むクラスタ番号を指定)
  - $\mathrm{low}(x) = x \mod \sqrt{u}$: xの下半分のビット(クラスタ内でのxの番号)
  - $\mathrm{index}(h, l) = h\sqrt{u} + l$
    so that $x = \mathrm{index}(\mathrm{high}(x), \mathrm{low}(x))$

$proto\text{-}vEB$ での操作

Member(V, x)(再掲)

```
if V.u == 2
    return V.A[x]
else
    return Member(V.cluster[high(x)], low(x))
```

注意：Memberにはsummaryは不要.

計算量は

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

Minimum(V)(再掲)

```
if V.u == 2
    if V.A[0] == 1
        return 0
    else if V.A[1] == 1
        return 1
    else
        return NIL
else
    mincluster = Minimum(V.summary)
    if mincluster = NIL             // 全てのビットが落ちていたらNIL
        return NIL
    else                            // どこか立っていたらそこでのMinimumをとる
        offset = Minimum(V.cluster[mincluster])
        return index(mincluster, offset)
```

最悪計算量は

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

$$T(2^{2^k}) = 2T(2^{2^{k-1}}) + O(1) = O(2^k) = O(\log u)$$

Successor(V, x)

```
if V.u == 2
    if x==0 and V.A[1]==1
        return 1
    else
        return NIL
else
    offset = Successor(V.cluster[high(x)], low(x)) // ここで再帰
    if offset != NIL
        return index(high(x), offset)
    else
        succ = Successor(V.summary, high(x))     // ここで再帰
        if succ != NIL
            offset = Minimum(V.cluster[succ])    // O(logu)
            return index(succ, offset)
        else
            return NIL
```

最悪計算量$T(u)$は$T(u) = 2T(\sqrt{u}) + O(\log u)$なので、$S(k) = T(2^{2^k})$とおいて

$$T(u) = S(k) = 2S(k-1) + O(2^k) = O(k2^k) = O(\log u \log\log u)$$

Insert(V, x)

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

計算量

$$T(u) = O(\log u)$$

Delete(V, x)(教科書では省略)

```
if V.u == 2
    V.A[x] = 0
else
    Delete(V.cluster[high(x)], low(x)) // ここで再帰
    if isEmpty(V.cluster[high(x)])  // O(log log u)
        Delete(V.summary, high(x))  // ここで再帰
```

isEmpty(V)

```
if V.u == 2
    return V.A[0] or V.A[1]
else
    return isEmpty(V.summary)
```

最悪計算量 $T(u) = O(\log u)$

要素の個数を保持しておいてInsertを適切に変更してもよい

ここまでproto-vEB

ここからvEB

記号の準備

- $u = 2^k$
  - $\sqrt[\uparrow]{u} = 2^{\lceil k/2 \rceil}$, $\sqrt[\downarrow]{u} = 2^{\lfloor k/2 \rfloor}$
    so that $u = \sqrt[\uparrow]{u}\sqrt[\downarrow]{u}$
- $x \in \mathbb{N}$
  - $\mathrm{high}(x) = \lfloor x / \sqrt[\downarrow]{u} \rfloor$
  - $\mathrm{low}(x) = x \mod \sqrt[\downarrow]{u}$
  - $\mathrm{index}(h, l) = h\sqrt[\downarrow]{u} + l$
    so that $x = \mathrm{index}(\mathrm{high}(x), \mathrm{low}(x))$

vEBの定義

- $vEB(2) = \{min, max\},\ (min, max \in \{0, 1\})$
- $vEB(u) =$
  - min, max : 保持している集合の最小/最大値
  - $summary : vEB(\sqrt[\uparrow]{u})$のポインタ
    (木のノードに対応)
  - $cluster[0..\sqrt[\uparrow]{u}] : vEB(\sqrt[\downarrow]{u})$のポインタの配列
    (各ポインタが子ノードに対応)

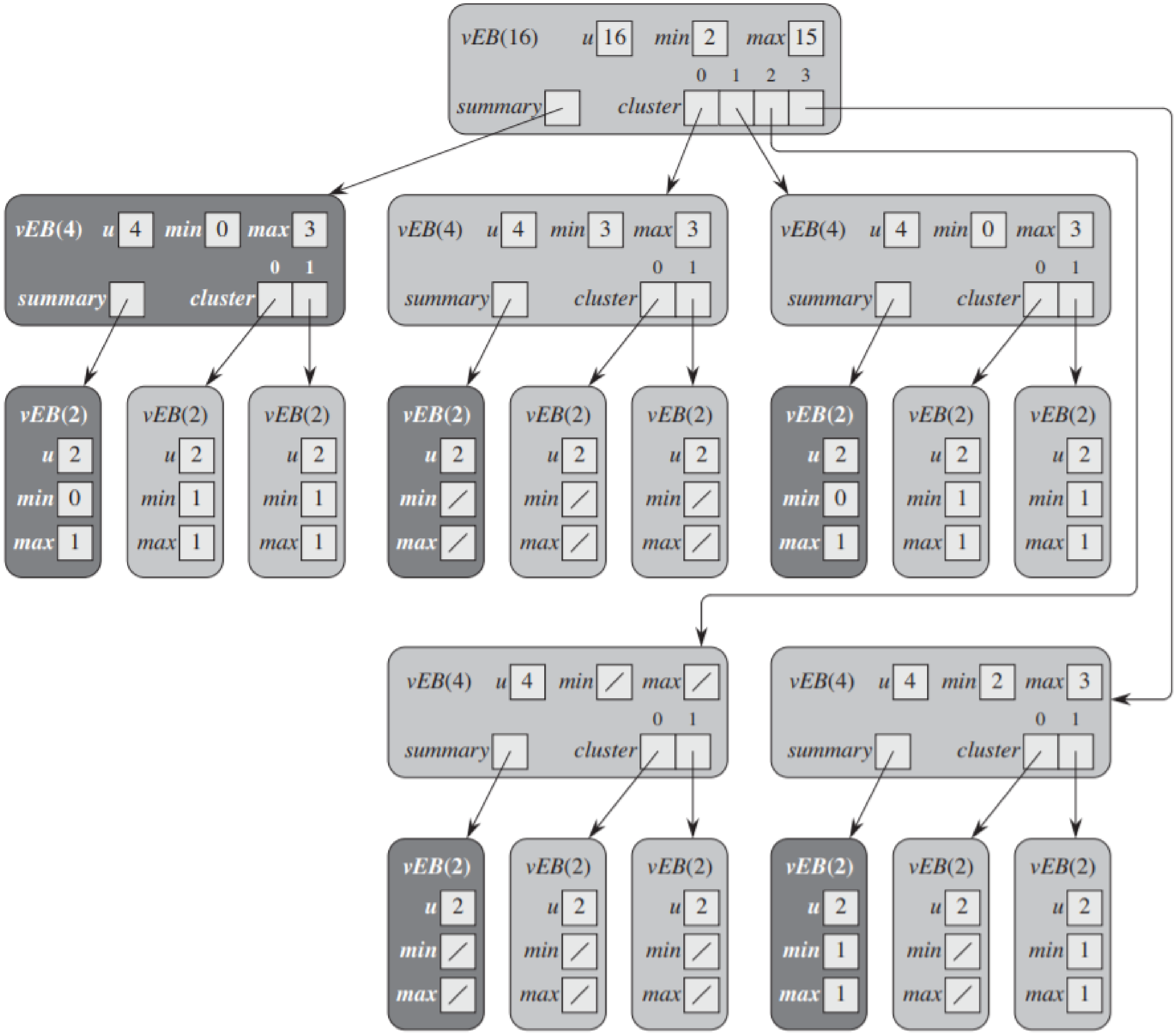
このとき、min**は子ノードに含まれない**ように構成する.
(Insert/Deleteで効いてくる)

ちょっと数学的に書くと

- for a vEB tree V, $A(V) := V$の保持する集合

  としたときに
  - $A(V) = \{V.min\} \cup \bigcup_{i=0}^{\uparrow\sqrt{u}} A(V.cluster[i])$ が非交和
  - $A(V.cluster[i]) \subset \{\mathrm{index}(i, 0), \cdots, \mathrm{index}(i+1, 0) - 1\} \cap A(V)$
  - $V.min = \min A(V)$
  - $V.max = \max A(V)$

を満たす.


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

$vEB(u)$における操作

Member(V, x)

```
if x == V.min or x == V.max
    return true
else if V.u == 2
    return false
else
    return Member(V.cluster[high(x)], low(x))
```

計算量は$O(\log \log u)$

Minimum/Maximum(V)

```
return V.min
```

```
return V.max
```

Successor(V, x)

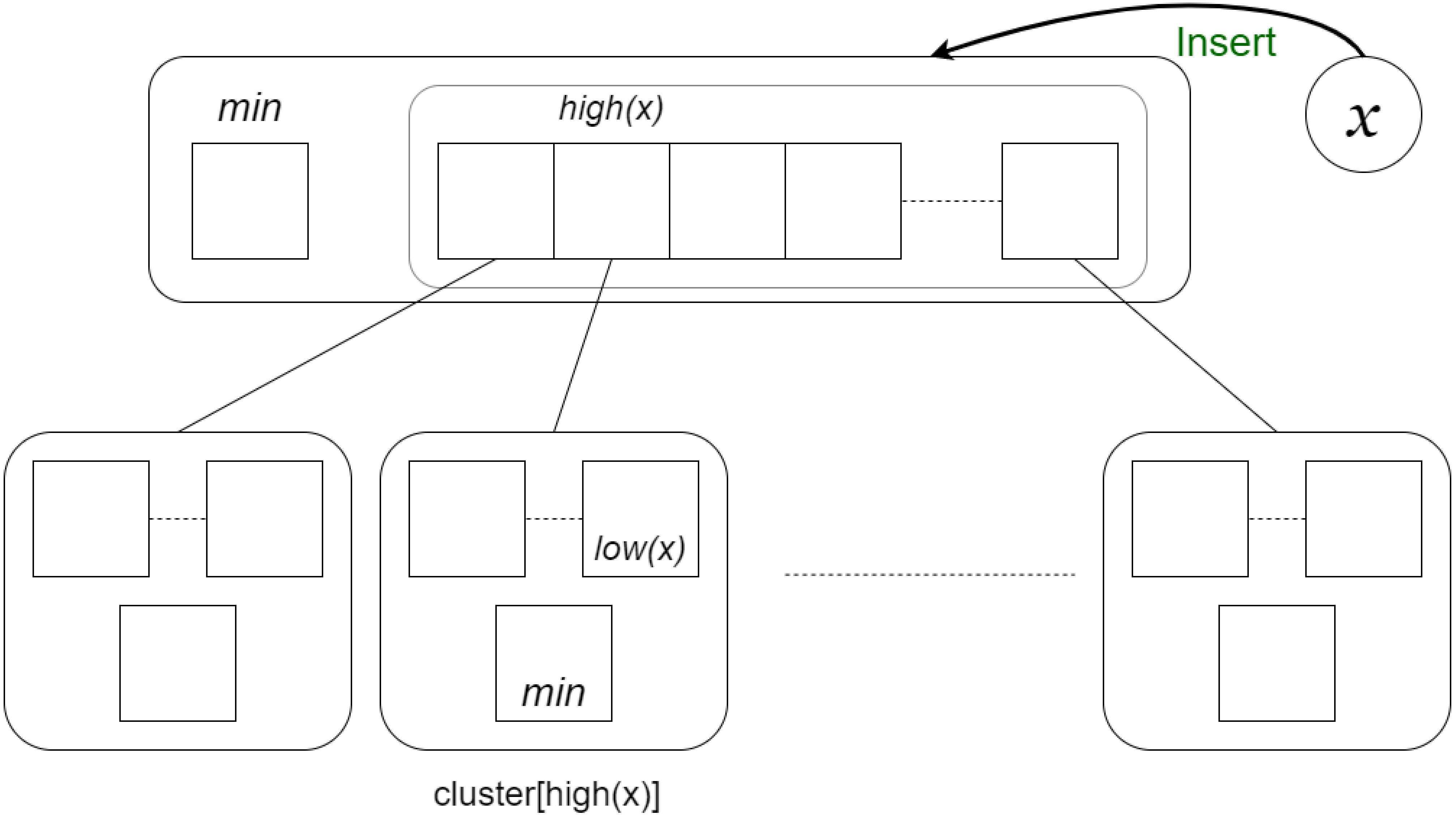
```
if V.u == 2
    if x == 0 and V.max == 1
        return 1
    else
        return NIL
else if V.min != NIL and x < V.min  // x がVに入っておらずminが返る場合
    return V.min
else
    max_low = Maximum(V.cluster[high(x)])
    if max_low != NIL and low(x) < max_low  // 同じクラスタ内にあるか判定
        offset = Successor(V.cluster[high(x)], low(x))
        return index(high(x), offset)
    else                                    // なければ次以降のクラスタのminimum
        succ = Successor(V.summary, high(x))
        if succ == NIL
            return NIL
        else
            return Minimum(V.cluster[succ])
```

再帰が高々1回になり、Minimumが$O(1)$になったので計算量は$O(\log \log u)$

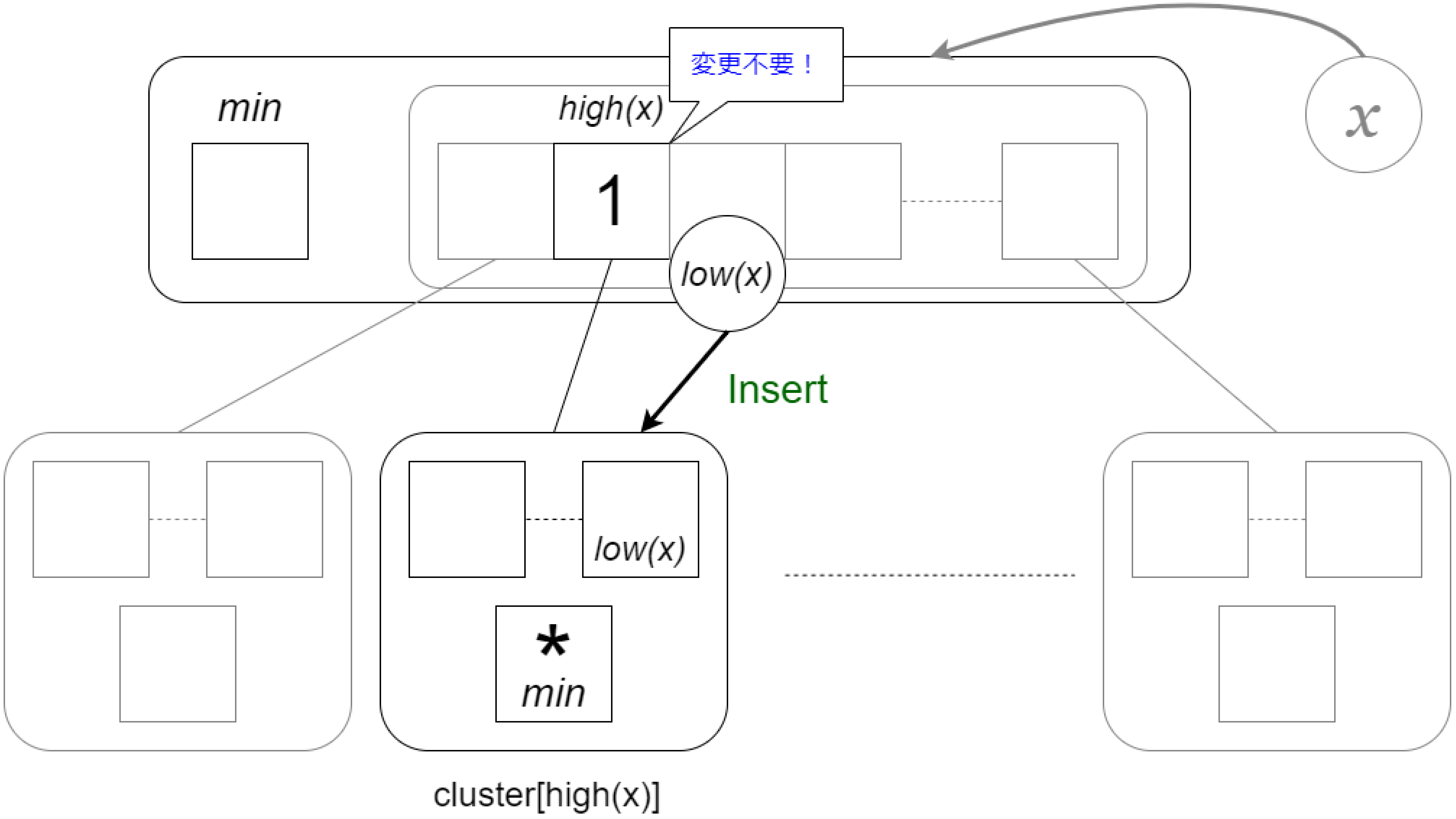
Predeccessor(V, x)

```
if V.u == 2
    if x == 1 and V.min == 0
        return 0
    else return NIL
else if V.max != NIL and V.max < x  // 論理的には不要？
    return V.max
else
    min_low = Minimum(V.cluster[high(x)])
    if min_low != NIL and low(x) > min_low
        offset = Successor(V.cluster[high(x)], low(x))
        return index(high(x), offset)
    else
        pred = Predecessor(V.summary, high(x))
        if pred == NIL
            if V.min != NIL and V.min < x   // Successorと異なる箇所
                return V.min
            else return NIL
        else
            return Maximum(V.cluster[pred])
```

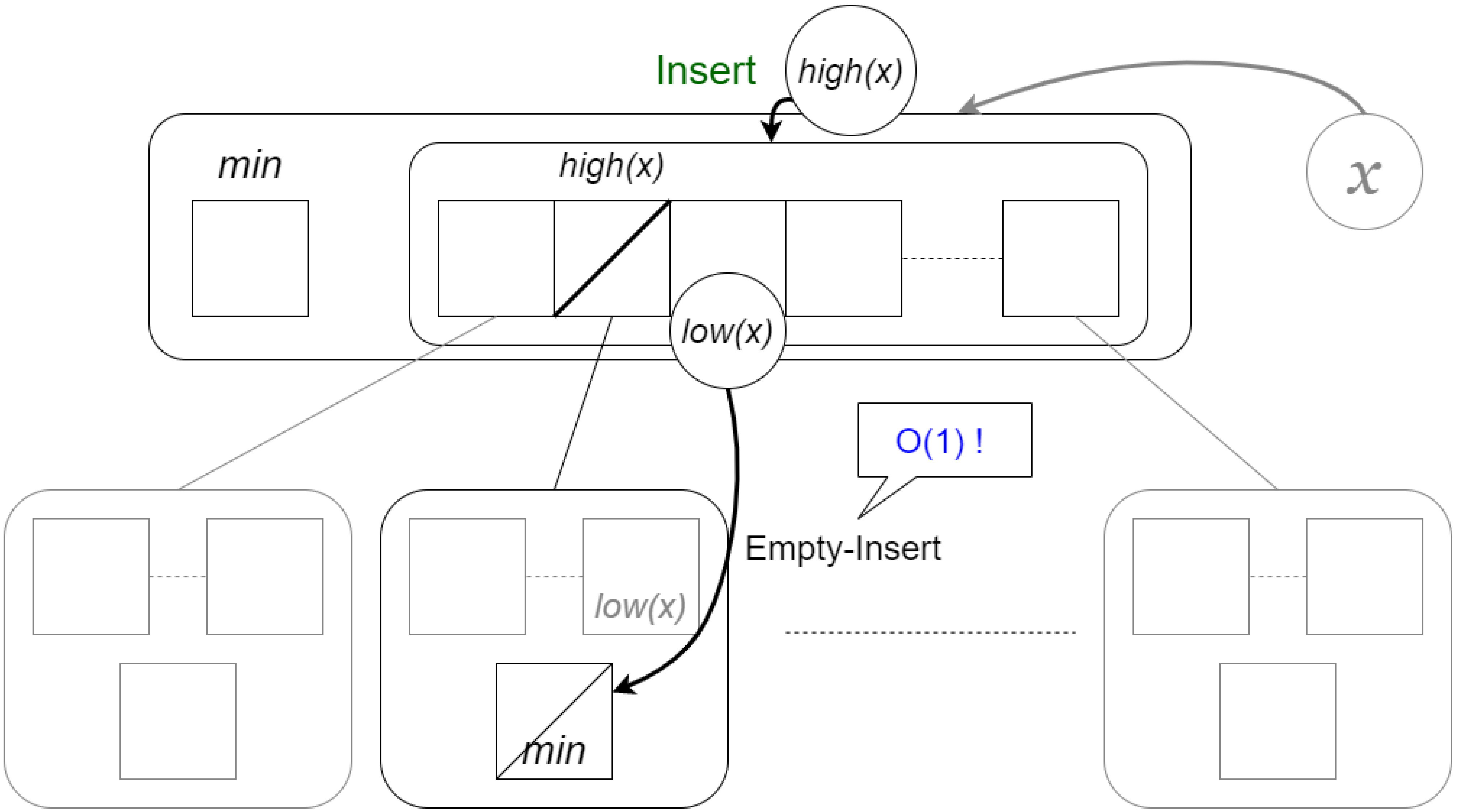
Insertの図



cluster[(high(x))]が空でないとき



cluster[(high(x))]が空のとき



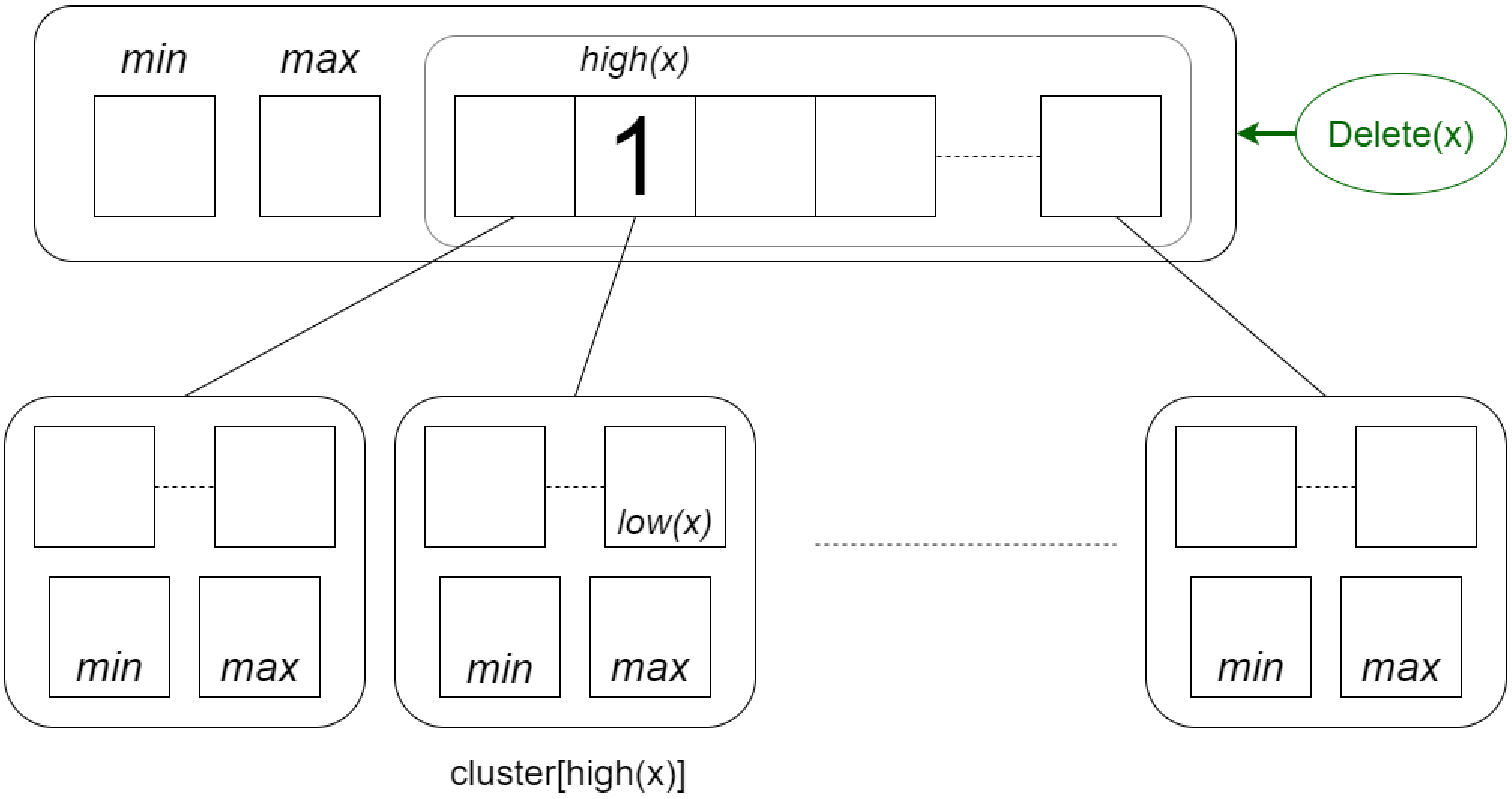
EmptyInsert(V, x)

```
V.min = x
V.max = x
```

Insert(V, x)

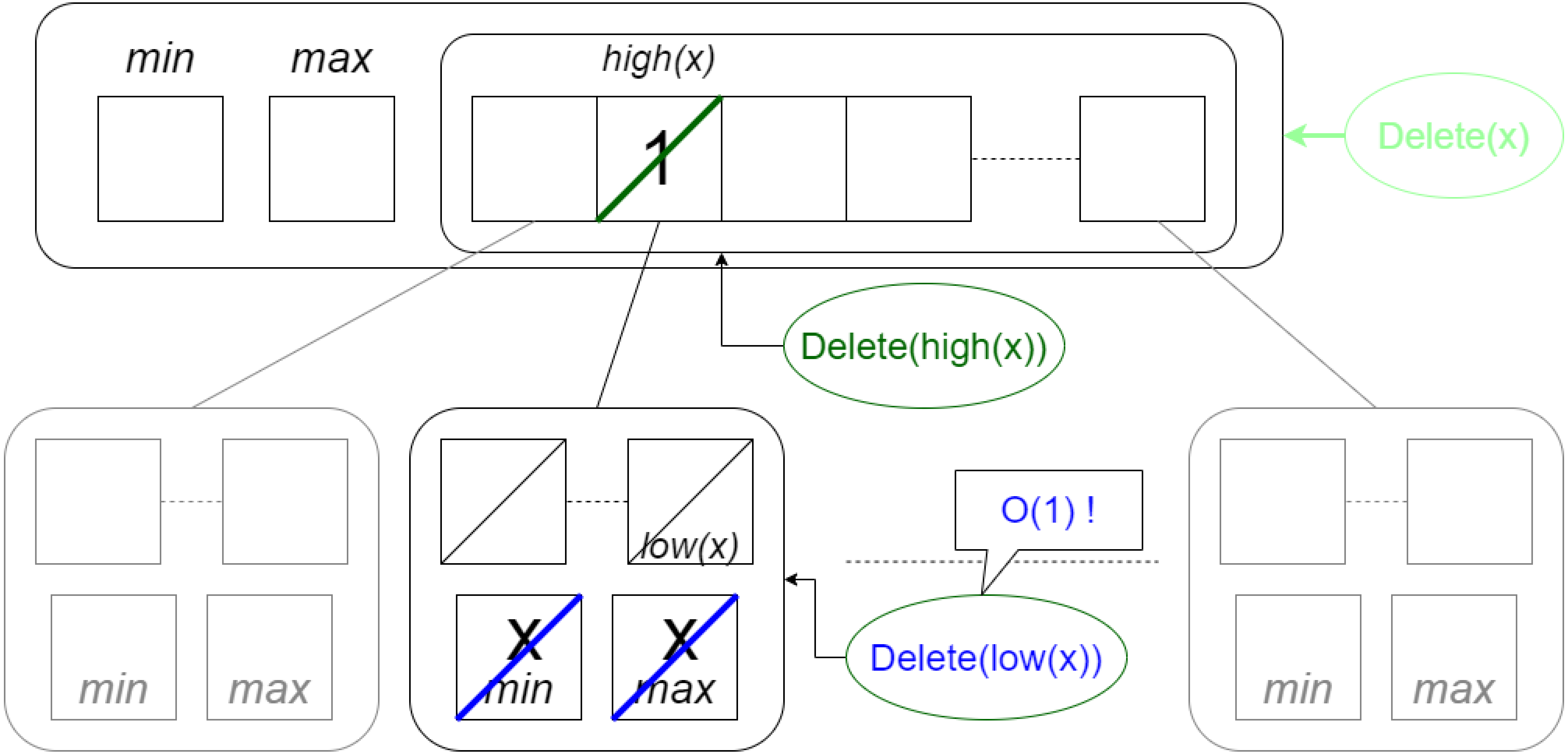
```
if V.min == NIL
    EmptyInsert(V, x)
else
    if x < V.min    // 最小値になるケースはそうでないケースに帰着
        temp = x
        x = V.min
        V.min = temp
    if V.u > 2
        if Minimum(V.cluster[high(x)]) == NIL
            EmptyInsert(V.cluster[high(x)], low(x))
            Insert(V.summary, high(x))
        else
            Insert(V.cluster[high(x)], low(x))
    if x > V.max    // maxの更新を忘れずに. u=2の場合もこれでカバーできる
        V.max = x
```

Deleteの図



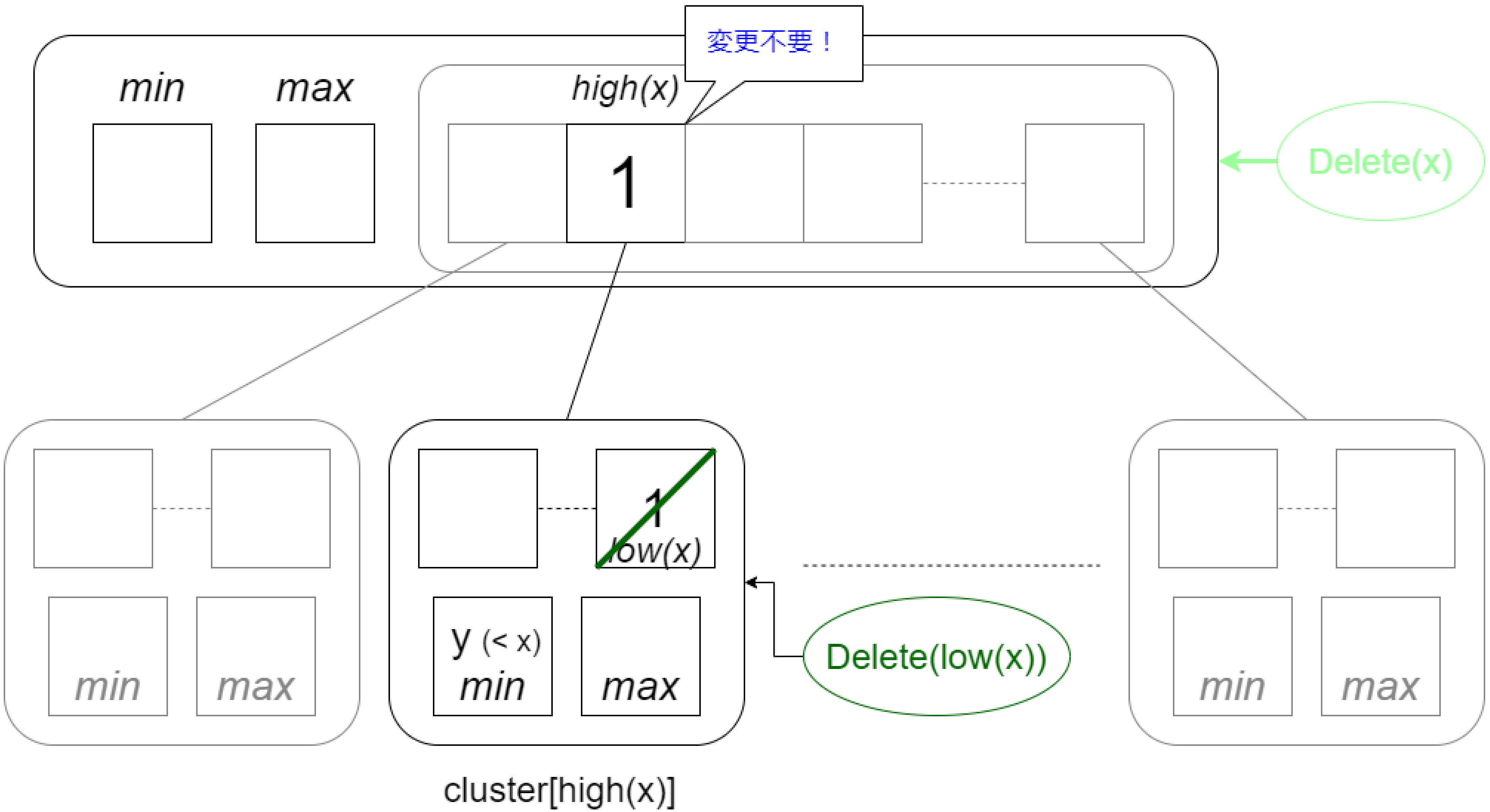
min
max
high(x)
1
Delete(x)
low(x)
min
max
min
max
min
max
cluster[high(x)]

cluster[(high(x))]が空になるとき



cluster[high(x)]

cluster[(high(x))]が空にならないとき



Delete(V, x)

```
if V.min == V.max
    V.min = NIL
    V.max = NIL
else if V.u == 2
    if x == 1
        V.min = 0
    else V.min = 1
    V.max = V.min
else
    if x == V.min   // 最小値を消す場合は上書きで削除、つじつま合わせは全く同様になる
        sum_min = Minimum(V.summary)
        x = index(sum_min, Minimum(V.cluster[sum_min]))
        V.min = x
    Delete(V.cluster[high(x)], low(x))
    if Minimum(V.cluster[high(x)]) == NIL
        Delete(V.summary, high(x))
        // （つづく）
```

Delete(V, x)(つづき)

```
    // max の更新
    (if Minimum(V.cluster[high(x)]) == NIL)
        if V.max == x
            sum_max = Maximum(V.summary)
            if sum_max == NIL
                V.max == V.min
            else
                V.max = index(sum_max, Maximum(V.cluster[sum_max]))
    else if V.max == x
        V.max = index(high(x), Maximum(V.cluster[high(x)]))
```

図より、計算量はInsertもDeleteも$O(\log \log u)$

おしまい