# van Emde Boas Trees

光吉　健汰

北海道大学工学部 情報エレクトロニクス学科 情報理工学コース 4 年
情報知識ネットワーク研究室

June 26, 2019

# Contents

# Contents

こんにちは

# Contents

# van Emde Boas tree とは (1/2)

van Emde Boas tree (以下 vEB 木) は動的集合を扱うデータ構造

**動的集合**

動的集合とは, 集合に対して後述の MEMBER$(V, x)$, INSERT$(V, x)$, DELETE$(V, x)$ が行えるようなデータ構造

- 今回扱う要素は非負整数とする
- vEB 木が保持しうる要素の集合を全体集合 $U$ とし, その大きさを $u$ とする
- vEB 木が現在保持している集合を $V$ とし, その大きさを $n$ とする

# van Emde Boas tree とは (2/2)

## 操作

$\textsc{Member}(V, x)$ : $V$ に $x$ が存在するかを返す

$\textsc{Min}(V)$ : $V$ の要素の最小値を返す

$\textsc{Max}(V)$ : $V$ の要素の最大値を返す

$\textsc{Successor}(V, x)$ : $V$ の $x$ より大きい最小の要素を返す

$\textsc{Predecessor}(V, x)$ : $V$ の $x$ より小さい最大の要素を返す

$\textsc{Insert}(V, x)$ : $V$ に $x$ を挿入する

$\textsc{Delete}(V, x)$ : $V$ から $x$ を削除する

# van Emde Boas tree とは (1/2)

1. 自己紹介
2. van Emde Boas tree とは
3. Binary tree
4. Proto van Emde Boas structures
5. van Emde Boas tree

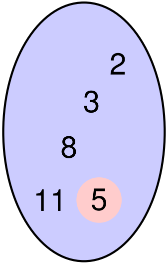
# van Emde Boas tree とは (2/2)

1. 自己紹介
2. van Emde Boas tree とは
3. Binary tree
4. Proto van Emde Boas structures
5. van Emde Boas tree

# 操作: $\textsc{Member}(V, x)$

- $\textsc{Member}(V, x)$ は, $x$ が集合 $V$ に存在するかを真偽値で返す.

$V = \{2, 3, 5, 8, 11\}$




- $\textsc{Member}(V, 5) = \text{TRUE}$
- $\textsc{Member}(V, 6) = \text{FALSE}$

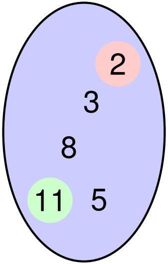
**各関数の引数**

関数に渡す引数は $U$ の要素のみ

# 操作: $\textsc{Min}(V)$, $\textsc{Max}(V)$

- $\textsc{Min}(V)$ は, 集合 $V$ の要素の最小値を返す.
- $\textsc{Max}(V)$ は, 集合 $V$ の要素の最大値を返す.

$V = \{2, 3, 5, 8, 11\}$

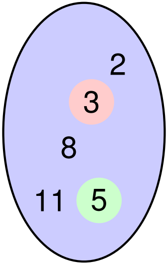



- $\textsc{Min}(V) = 2$
- $\textsc{Max}(V) = 11$

# 操作: $\textsc{Successor}(V, x)$, $\textsc{Predecessor}(V, x)$

- $\textsc{Successor}(V, x)$ は, 集合 $V$ 中の $x$ より大きい最小の要素を返す.
- $\textsc{Predecessor}(V, x)$ は, 集合 $V$ 中の $x$ より小さい最大の要素を返す.

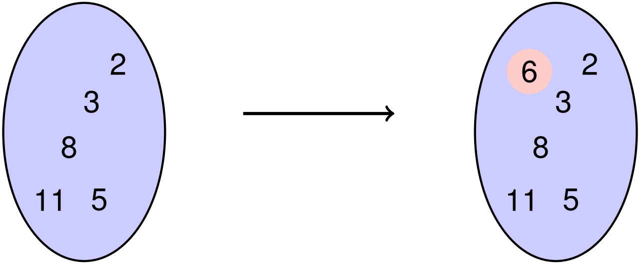
$V = \{2, 3, 5, 8, 11\}$




- $\textsc{Successor}(V, 2) = \boxed{3}$
- $\textsc{Predecessor}(V, 7) = \boxed{5}$

- Insert$(V, x)$ は, 集合 $V$ に $x$ を挿入する.
  - $V \leftarrow V \cup \{x\}$

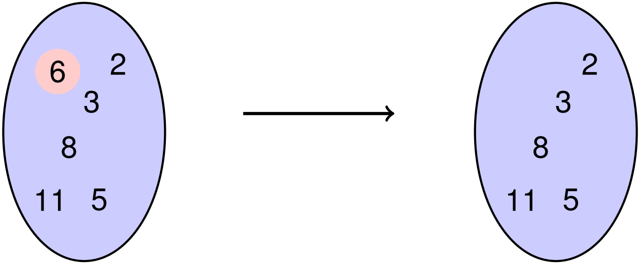
$V = \{2, 3, 5, 8, 11\}$ $\qquad$ $V = \{2, 3, 5, \mathbf{6}, 8, 11\}$



# 操作: DELETE$(V, x)$

- DELETE$(V, x)$ は, 集合 $V$ から $x$ を削除する.
  - $V \leftarrow V \setminus \{x\}$

$V = \{2, 3, 5, \mathbf{6}, 8, 11\}$ $V = \{2, 3, 5, 8, 11\}$



# Contents

# 直接アドレス法 (1/2)

動的集合を保持する手段として, 直接アドレス法を考える.

## 直接アドレス法

要素の値を配列の添字として利用し, データを保持するテクニック

今回考えているデータ構造では
ハッシュ表のような付属データは持たないので,
各配列の要素には要素を保持しているかを bit で格納する.

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

$$V = \{2, 3, 5, 8, 11\}$$

# 直接アドレス法 (2/2)

- MEMBER$(V, x)$, INSERT$(V, x)$, DELETE$(V, x)$ は,
配列のランダムアクセスが定数時間なので, 時間計算量 $O(1)$.
- SUCCESSOR$(V, x)$ は, $x$ の次の値が 1 の要素まで
最悪 $\Theta(u)$ 回探索する必要があるので, 時間計算量 $O(u)$.
    - PREDECESSOR$(V, x)$, MIN$(V)$, MAX$(V)$ も同様の処理を行うため,
    時間計算量 $O(u)$.

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

$$V = \{2, 3, 5, 8, 11\}$$

# 二分木構造 (1/3)

$\textsc{Successor}(V, x)$ で必要な探索回数を小さくするために,
配列の各要素を葉とした二分木構造を考える.

## 二分木

二分木とは, 葉ではない頂点の子が常に 2 個であるような木

木 : 閉路を持たない, 連結なグラフ

根 : 木の頂点のうちの 1 つを定義する

子 : ある頂点について, 隣接する頂点のうち根から遠いもの

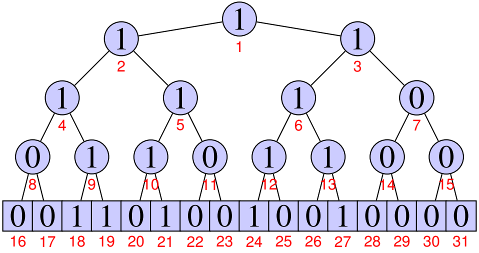
親 : ある頂点について, 隣接する頂点のうち根に近いもの

葉 : 木の頂点のうち, 子を持たないもの

# 二分木構造 (2/3)

葉ではない各頂点には, 子の値の論理和を格納する.

- ある頂点の値が 1 の場合,
  その頂点の下にある葉の少なくとも 1 つの頂点は値が 1.
- 根から深さ毎に左から配列に詰めて保持することで
  各頂点にランダムアクセスが可能となる.
  - 赤字は格納されている配列の添字 (後述の操作のために 1-*indexed* とする)

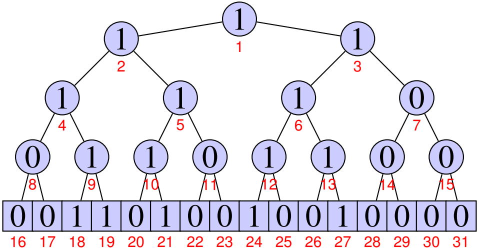



$$V = \{2, 3, 5, 8, 11\}$$

# 二分木構造 (3/3)

二分木中のある頂点の添字を $i$ とすると,
以下のように他の頂点の添字を求めることができる.

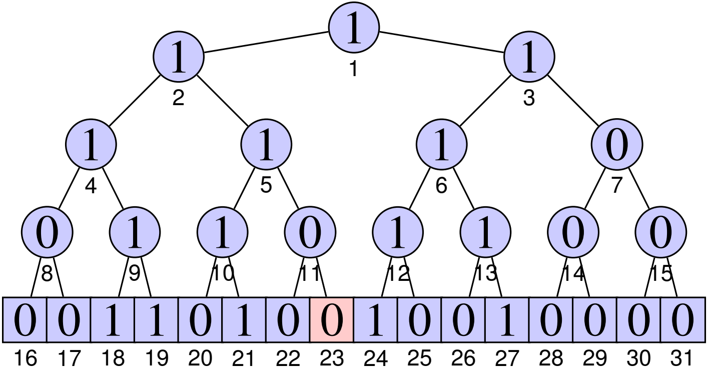
- **根** : 先頭なので 1
- **子** : $2 \times i + 1$ と $2 \times i + 2$
- **親** : $\lfloor \frac{i}{2} \rfloor$ (以後 $\frac{r}{m}$ が整数でない場合これを切り捨てた値とする)
- **葉** : 集合 $U$ の要素 $v$ に対応する葉の添字は $i + u$




$$V = \{2, 3, 5, 8, 11\}$$

# 二分木構造 操作: MEMBER($V, x$)

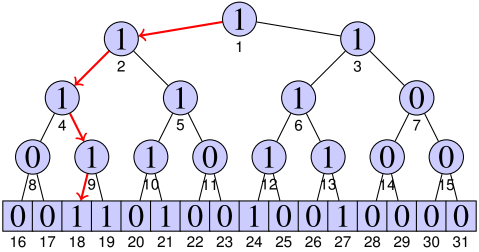
- MEMBER($V, x$) は, 対応する葉の値を返す.




MEMBER($V, 7$), $V = \{2, 3, 5, 8, 11\}$

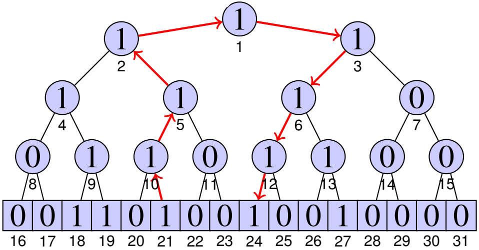
# 二分木構造 操作: $\textsc{Min}(V)$, $\textsc{Max}(V)$

- $\textsc{Min}(V)$ は, 根から左の子が $1$ であれば左の子へ, そうでなければ右の子へ降りる操作を再帰的に行う.
  - $\textsc{Max}(V)$ も同様に処理する.



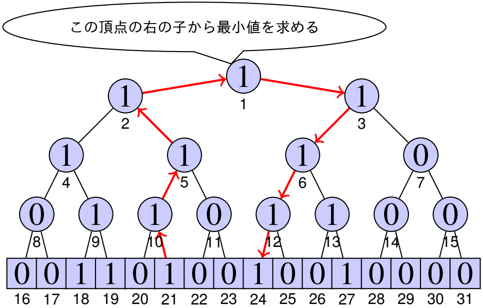

$\textsc{Min}(V), V = \{2, 3, 5, 8, 11\}$

- Successor$(V, x)$ は, $x$ の親から, 自身が直前の頂点が左の子かつ右の子の値が 1 の頂点になるまで親を辿り,
  右の子の最小値を返す.
  - Predecessor$(V, x)$ も同様に処理する.




Successor$(V, 5)$, $V = \{2, 3, 5, 8, 11\}$

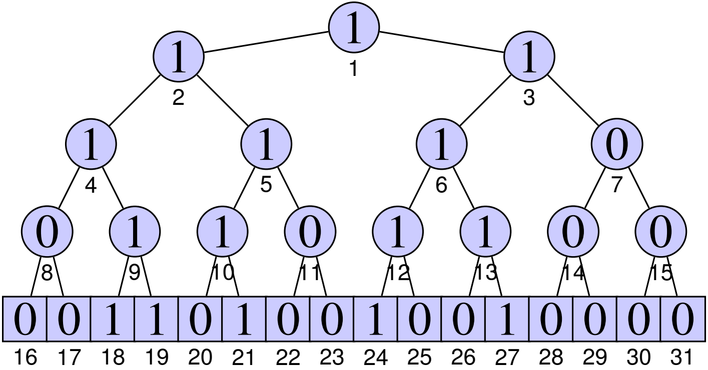
- Successor$(V, x)$ は, $x$ の親から, 自身が直前の頂点が左の子かつ右の子の値が $1$ の頂点になるまで親を辿り,
  右の子の最小値を返す.
  - Predecessor$(V, x)$ も同様に処理する.




Successor$(V, 5)$, $V = \{2, 3, 5, 8, 11\}$

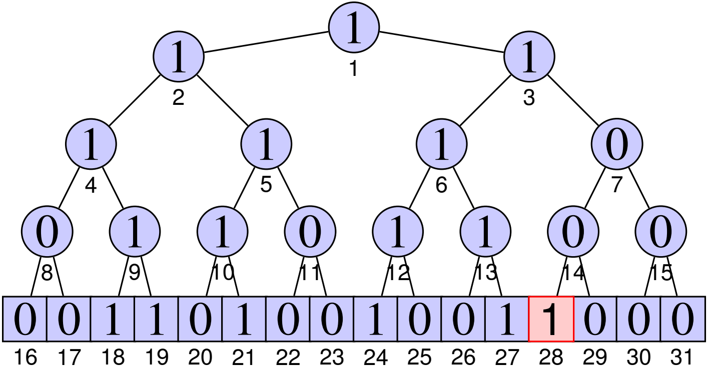
# 二分木構造 操作: Insert$(V, x)$

- Insert$(V, x)$ は, 対応する葉から親の値を 1 にしつつ根まで辿る.




Insert$(V, 12)$, $V = \{2, 3, 5, 8, 11\}$

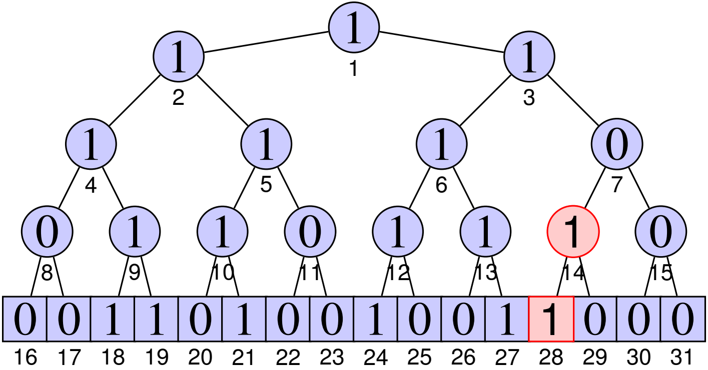
- INSERT$(V, x)$ は, 対応する葉から親の値を 1 にしつつ根まで辿る.




INSERT$(V, 12)$, $V = \{2, 3, 5, 8, 11\}$

# 二分木構造 操作: Insert($V, x$)

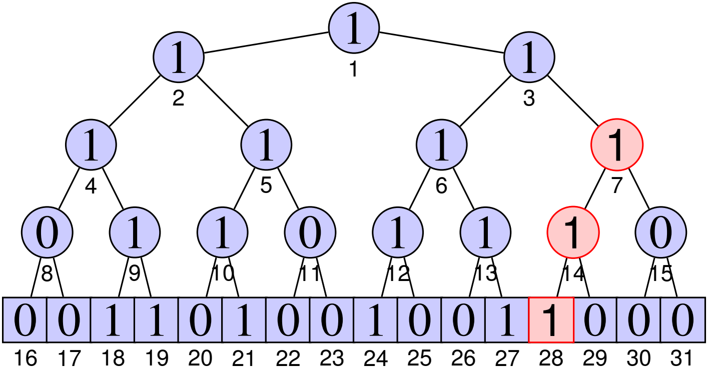
- Insert($V, x$) は, 対応する葉から親の値を 1 にしつつ根まで辿る.




Insert($V$, 12), $V = \{2, 3, 5, 8, 11\}$

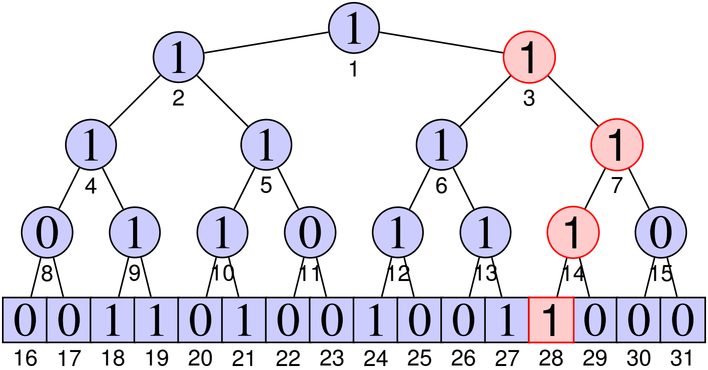
# 二分木構造 操作: INSERT($V, x$)

- INSERT($V, x$) は, 対応する葉から親の値を 1 にしつつ根まで辿る.




INSERT($V, 12$), $V = \{2, 3, 5, 8, 11\}$

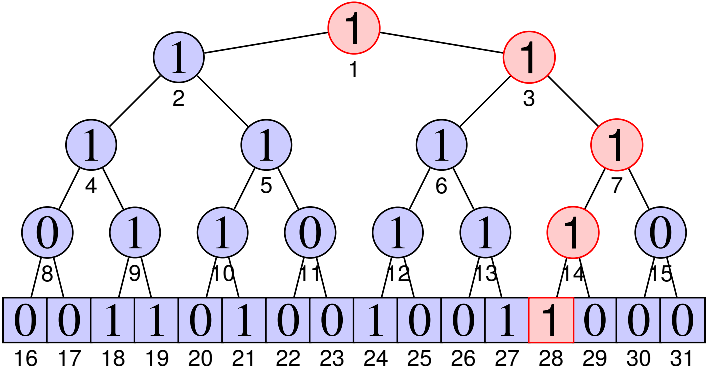
- INSERT$(V, x)$ は, 対応する葉から親の値を 1 にしつつ根まで辿る.




INSERT$(V, 12)$, $V = \{2, 3, 5, 8, 11\}$

# 二分木構造 操作: Insert$(V, x)$

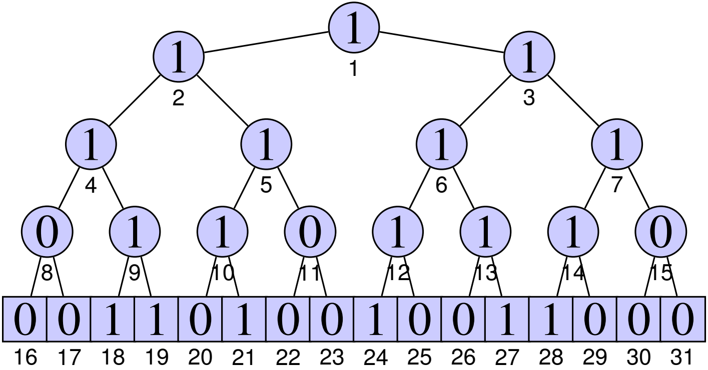
- Insert$(V, x)$ は, 対応する葉から親の値を $1$ にしつつ根まで辿る.




Insert$(V, 12), V = \{2, 3, 5, 8, 11\}$

# 二分木構造 操作: DELETE$(V, x)$

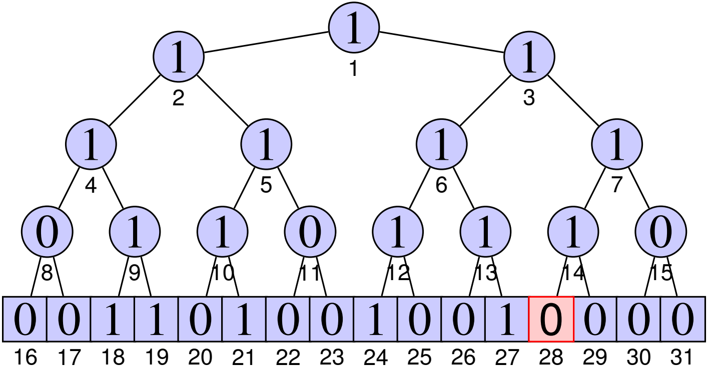
- DELETE$(V, x)$ は, 対応する葉の値を $0$ にした後, 以下の操作を行いながら根まで辿る.
  - 子が両方 $0$ ならば, 自身の値を $0$ にし, 親の頂点に遷移する.




DELETE$(V, 12), V = \{2, 3, 5, 6, 8, 11, 12\}$

# 二分木構造 操作: Delete($V, x$)

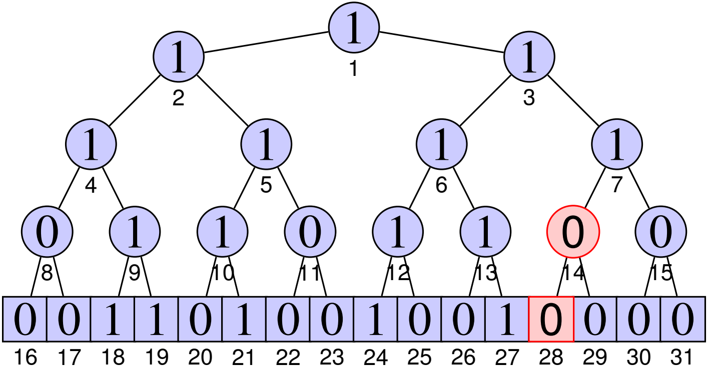
- Delete($V, x$) は, 対応する葉の値を $0$ にした後, 以下の操作を行いながら根まで辿る.
  - 子が両方 $0$ ならば, 自身の値を $0$ にし, 親の頂点に遷移する.




Delete($V, 12$), $V = \{2, 3, 5, 6, 8, 11, 12\}$

# 二分木構造 操作: DELETE$(V, x)$

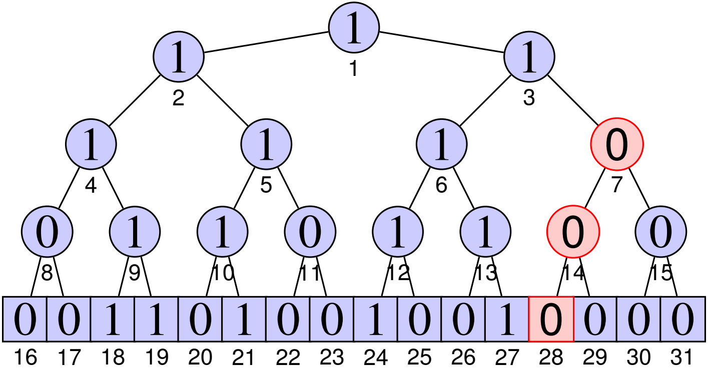
- DELETE$(V, x)$ は, 対応する葉の値を 0 にした後, 以下の操作を行いながら根まで辿る.
  - 子が両方 0 ならば, 自身の値を 0 にし, 親の頂点に遷移する.




DELETE$(V, 12), V = \{2, 3, 5, 6, 8, 11, 12\}$

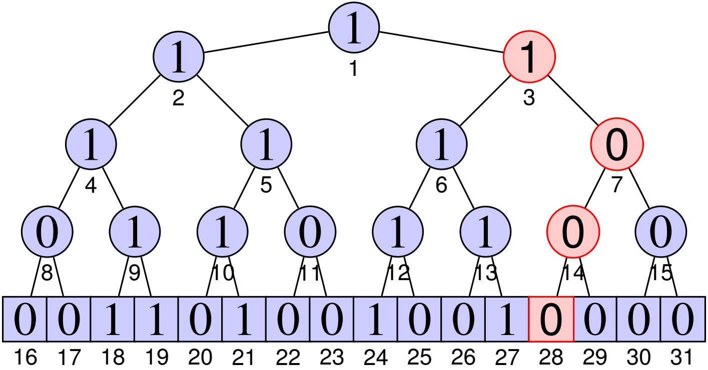
# 二分木構造 操作: DELETE($V, x$)

- DELETE($V, x$) は, 対応する葉の値を 0 にした後, 以下の操作を行いながら根まで辿る.
  - 子が両方 0 ならば, 自身の値を 0 にし, 親の頂点に遷移する.



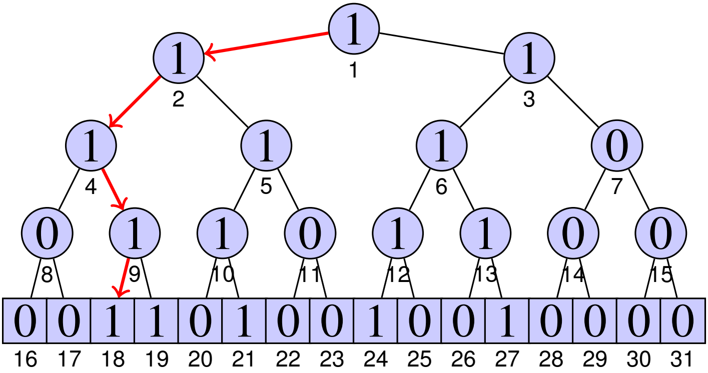

DELETE($V$, 12), $V = \{2, 3, 5, 6, 8, 11, 12\}$

- DELETE$(V, x)$ は, 対応する葉の値を $0$ にした後, 以下の操作を行いながら根まで辿る.
  - 子が両方 $0$ ならば, 自身の値を $0$ にし, 親の頂点に遷移する.



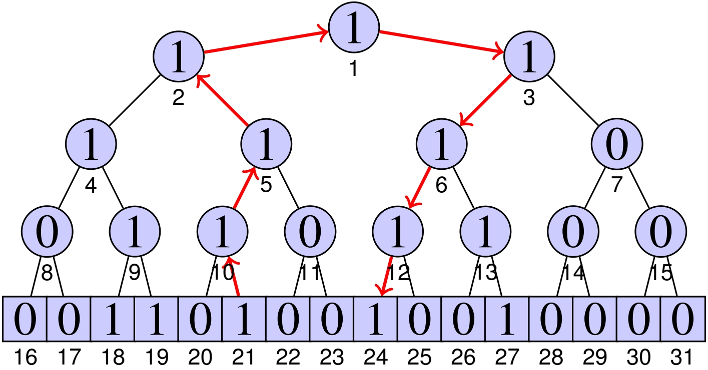

DELETE$(V, 12)$, $V = \{2, 3, 5, 6, 8, 11, 12\}$

- $\textsc{Min}(V)$, $\textsc{Max}(V)$ では, 根から最小値となる葉まで頂点を探索するので, 時間計算量は $\Theta(\log u)$.




$\textsc{Min}(V), V = \{2, 3, 5, 8, 11\}$

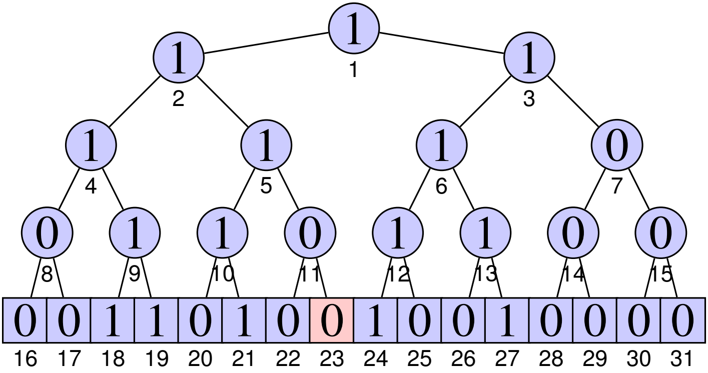
- Successor$(V, x)$, Predecessor$(V, x)$ では, 葉から最悪根まで辿り, その後, Min$(V)$ (または Max$(V)$) と同様の処理を行うことから, 時間計算量は $O(\log u)$.




Successor$(V, 5)$, $V = \{2, 3, 5, 8, 11\}$

# 二分木構造 計算量 (3/4)

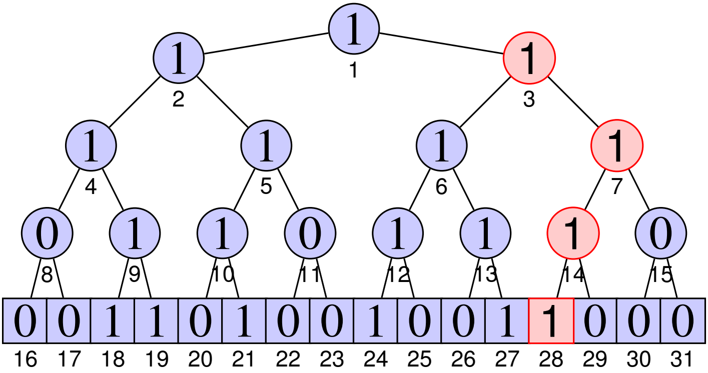
- $\textsc{Member}(V, x)$ では, 対応する葉にアクセスし, 値を取得する, もしくは更新するので, 時間計算量は $O(1)$.




$\textsc{Member}(V, 7), V = \{2, 3, 5, 8, 11\}$

# 二分木構造 計算量 (4/4)

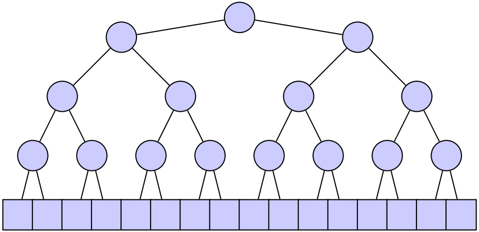
- Insert$(V, x)$, Delete$(V, x)$ では,
  葉から根まで辿る処理を行うので, 時間計算量は $O(\log u)$.




Insert$(V, 12)$, $V = \{2, 3, 5, 8, 11\}$

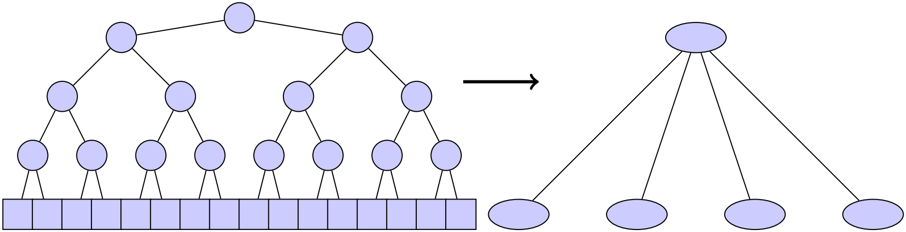
# 二分木構造 高速化 (1/2)

二分木構造では, 各種操作の時間計算量が $O(\log u)$ となる.



# 二分木構造 高速化 (2/2)

二分木構造では, 各種操作の時間計算量が $O(\log u)$ となる.
→ 木の高さに時間計算量が依存している.
頂点あたりの子を増やし, 木の高さを小さくする.



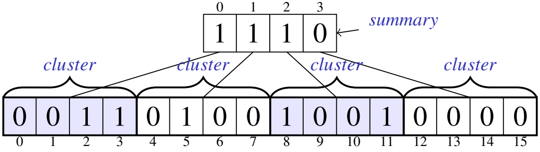
# 平方分割木

葉の数を $\sqrt{u}$ 個 とした平方分割木 (仮称) を考える.

## 平方分割木

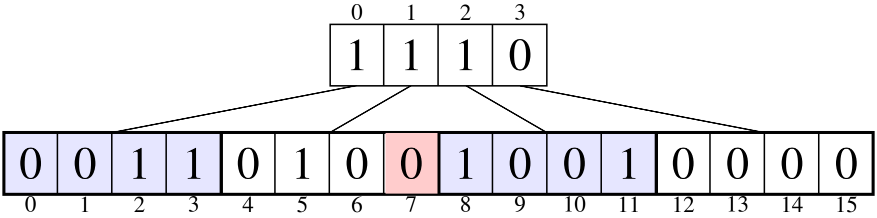
平方分割木とは, 葉を $\sqrt{u}$ 個持ち, 高さが 2 の,
以下のような特徴を持つデータ構造である.

- 要素の全体集合の大きさ $u$ を $u = 2^{2k}$($k$ は非負整数) とする.
- 配列を $\sqrt{u}$ 個に分割し, それぞれを *cluster* とする.
- 根は大きさ $\sqrt{u}$ の配列を持ち (これを *summary* とする) , それぞれの要素は各 *cluster* の要素の論理和を保持する.




$$V = \{2, 3, 5, 8, 11\}$$

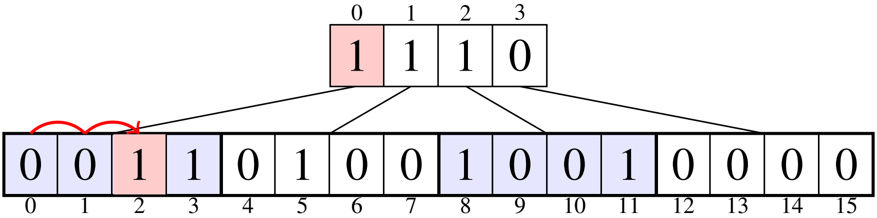
- MEMBER($V, x$) は, 対応する配列の要素の値を返す.




MEMBER($V$, 7), $V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: $\textsc{Min}(V)$, $\textsc{Max}(V)$

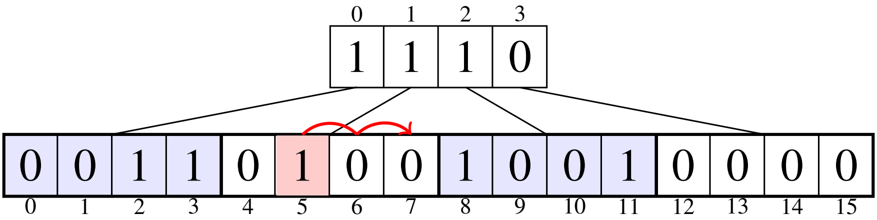
- $\textsc{Min}(V)$ は, *summary* で値が 1 の左端の要素を探し, 対応する *cluster* で最小値を左端から探す.
  - $\textsc{Max}(V)$ も同様に処理する.




$\textsc{Min}(V), V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: $\textsc{Successor}(V, x)$, $\textsc{Predecessor}(V, x)$

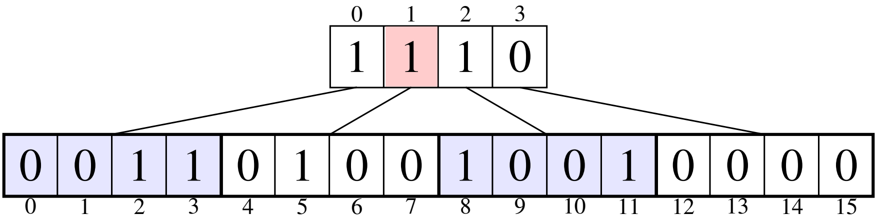
- $\textsc{Successor}(V, x)$ は, $x$ の *cluster* 内で $x$ より左の要素を探索し, もしなければ, *summary* で $\frac{x}{\sqrt{u}}$ より右で 1 である要素を探す. その後, 1 の値である *cluster* 内の最小値を返す.
  - $\textsc{Predecessor}(V, x)$ も同様に処理する.




$\textsc{Min}(V), V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: $\textsc{Successor}(V, x)$, $\textsc{Predecessor}(V, x)$

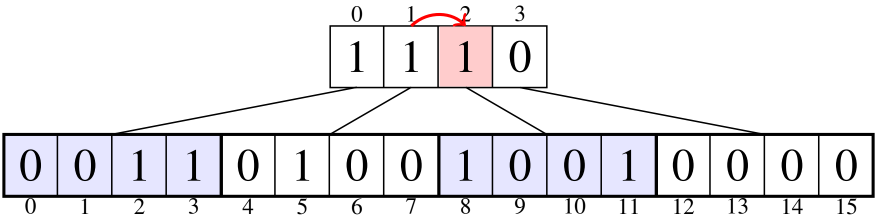
- $\textsc{Successor}(V, x)$ は, $x$ の *cluster* 内で $x$ より左の要素を探索し, もしなければ, *summary* で $\frac{x}{\sqrt{u}}$ より右で 1 である要素を探す. その後, 1 の値である *cluster* 内の最小値を返す.
  - $\textsc{Predecessor}(V, x)$ も同様に処理する.




$\textsc{Min}(V)$, $V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: Successor$(V, x)$, Predecessor$(V, x)$

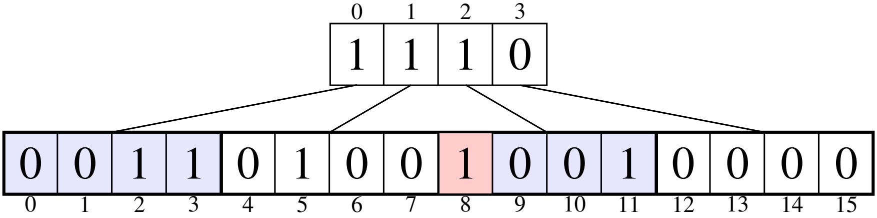
- Successor$(V, x)$ は, $x$ の *cluster* 内で $x$ より左の要素を探索し, もしなければ, *summary* で $\frac{x}{\sqrt{u}}$ より右で 1 である要素を探す. その後, 1 の値である *cluster* 内の最小値を返す.
    - Predecessor$(V, x)$ も同様に処理する.




Min$(V)$, $V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: Successor$(V, x)$, Predecessor$(V, x)$

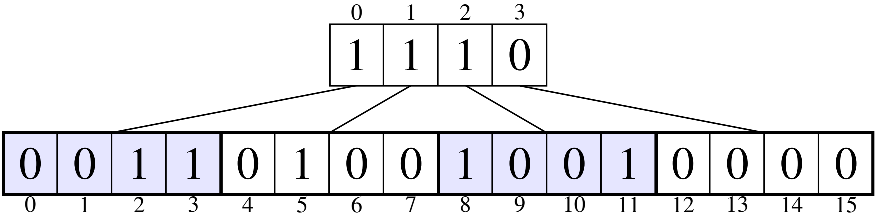
- Successor$(V, x)$ は, $x$ の *cluster* 内で $x$ より左の要素を探索し, もしなければ, *summary* で $\frac{x}{\sqrt{u}}$ より右で 1 である要素を探す. その後, 1 の値である *cluster* 内の最小値を返す.
  - Predecessor$(V, x)$ も同様に処理する.




Min$(V)$, $V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: Insert$(V, x)$

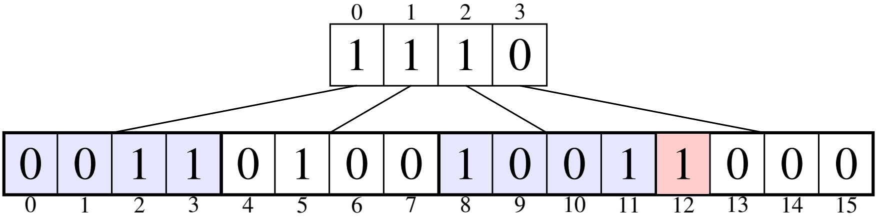
- Insert$(V, x)$ は, 対応する *summary*, *cluster* の要素を 1 にする.




Insert$(V, 12)$, $V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: $\textsc{Insert}(V, x)$

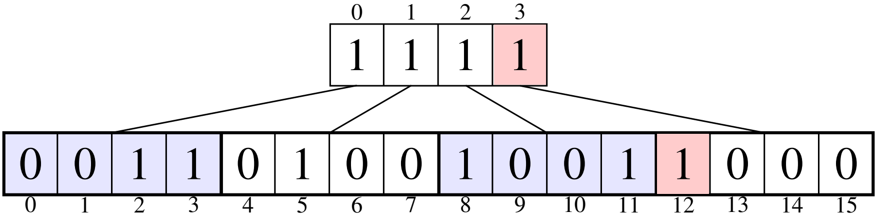
- $\textsc{Insert}(V, x)$ は, 対応する *summary*, *cluster* の要素を 1 にする.




$\textsc{Insert}(V, 12), V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: $\textsc{Insert}(V, x)$

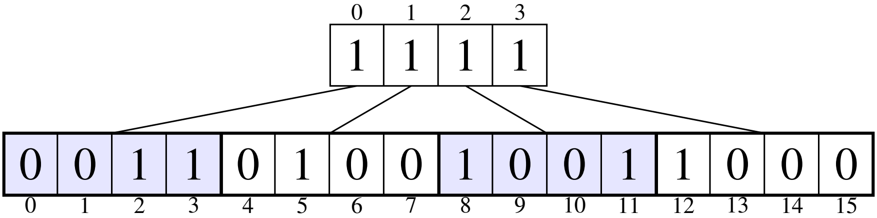
- $\textsc{Insert}(V, x)$ は, 対応する *summary*, *cluster* の要素を 1 にする.




$\textsc{Insert}(V, 12), V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造 操作: DELETE$(V, x)$

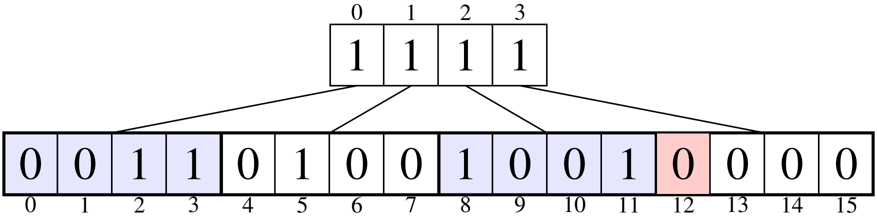
- DELETE$(V, x)$ は 対応する *cluster* の要素の値を 0 にし，*cluster* の全要素が 0 ならば，*summary* の要素を 0 にする．




DELETE$(V, 12), V = \{2, 3, 5, 8, 11, 12\}$

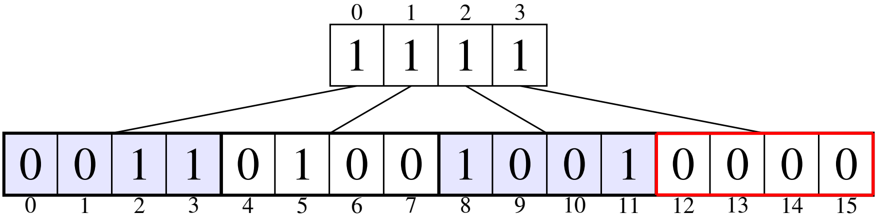
# 平方分割木構造 操作: DELETE($V, x$)

- DELETE($V, x$) は 対応する *cluster* の要素の値を 0 にし，*cluster* の全要素が 0 ならば，*summary* の要素を 0 にする．




DELETE($V$, 12), $V = \{2, 3, 5, 8, 11, 12\}$

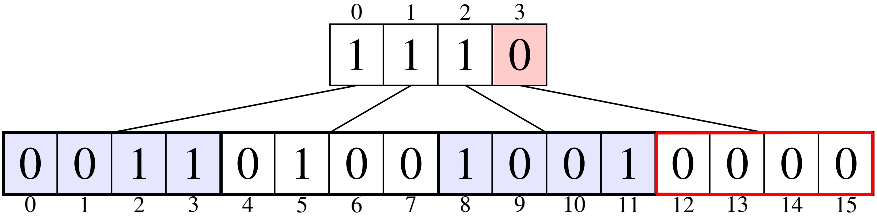
- DELETE$(V, x)$ は 対応する *cluster* の要素の値を 0 にし, *cluster* の全要素が 0 ならば, *summary* の要素を 0 にする.




DELETE$(V, 12), V = \{2, 3, 5, 8, 11, 12\}$

# 平方分割木構造 操作: DELETE$(V, x)$

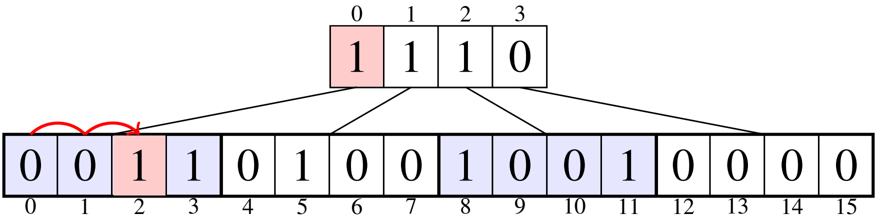
- DELETE$(V, x)$ は 対応する *cluster* の要素の値を 0 にし, *cluster* の全要素が 0 ならば, *summary* の要素を 0 にする.




DELETE$(V, 12)$, $V = \{2, 3, 5, 8, 11, 12\}$

# 平方分割木構造: 計算量 (1/4)

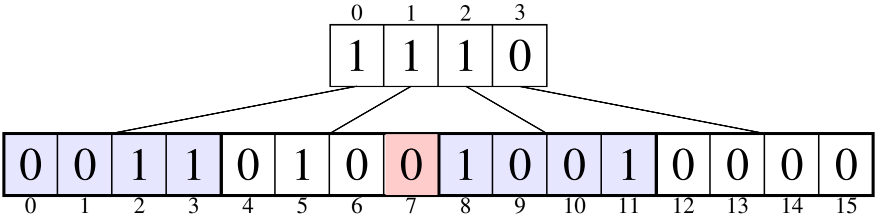
- $\textsc{Min}(V)$, $\textsc{Max}(V)$ では, *summary* と, *cluster*1 つを線形探索するので, 時間計算量は $O(\sqrt{u})$ となる.
- $\textsc{Successor}(V, x)$, $\textsc{Predecessor}(V, x)$ も $\textsc{Min}(V)$ と同様に時間計算量は $O(\sqrt{u})$ となる.




$\textsc{Min}(V), V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造: 計算量 (2/4)

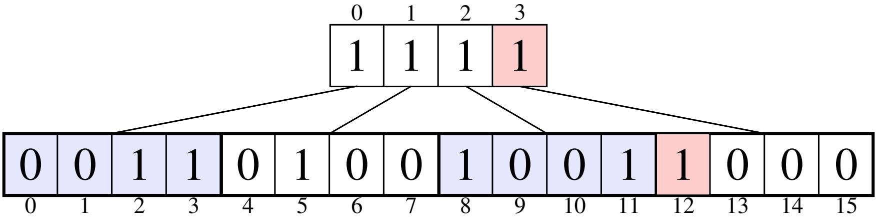
- $\textsc{Member}(V, x)$ では,
  対応する *cluster* の要素を取得すればよいので, 時間計算量は $O(1)$ となる.




$\textsc{Member}(V, 7), V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造: 計算量 (3/4)

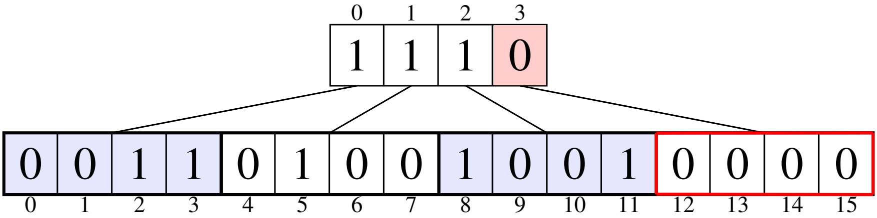
- $\textsc{Insert}(V, x)$ では,
  対応する *cluster*, *summary* の要素を更新すればよいので, 時間計算量は $O(1)$ となる.




$\textsc{Insert}(V, 12), V = \{2, 3, 5, 8, 11\}$

# 平方分割木構造: 計算量 (4/4)

- $\textsc{Delete}(V, x)$ では, 対応する *cluster* を線形探索するので, 時間計算量は $O(\sqrt{u})$ となる.




$\textsc{Delete}(V, 12), V = \{2, 3, 5, 8, 11, 12\}$

# 前半のまとめ

- vEB 木を定義した.
- 二分木構造で各種操作の時間計算量は下表となった.
- 平方分割木によって二分木構造の木の高さを小さくしたが,
  一部操作の最悪時間計算量が $O(\sqrt{u})$ と悪化
  - *summary*, *cluster* の線形探索がボトルネック

| | 二分木 | 平方分割木 |
|---|---|---|
| $\textsc{Min}(V)$ | $\Theta(\log u)$ | $O(\sqrt{u})$ |
| $\textsc{Max}(V)$ | $\Theta(\log u)$ | $O(\sqrt{u})$ |
| $\textsc{Member}(V, x)$ | $O(1)$ | $O(1)$ |
| $\textsc{Successor}(V, x)$ | $O(\log u)$ | $O(\sqrt{u})$ |
| $\textsc{Predecessor}(V, x)$ | $O(\log u)$ | $O(\sqrt{u})$ |
| $\textsc{Insert}(V, x)$ | $O(\log u)$ | $O(1)$ |
| $\textsc{Delete}(V, x)$ | $O(\log u)$ | $O(\sqrt{u})$ |

# Contents

# Contents