Van Emde Boas Stromy

→ množina klíčů = $\{0, 1, 2, \dots, u-1\}$,

→ u mocnina 2, tj. $u = 2^k$ pro $k \geq 1$.

→ operace: MEMBER(x), INSERT(x), DELETE(x), MIN, MAX, SUCCESSOR(x), PREDECESSOR(x)

x je klíč.

Lze použít pro prioritní frontu: extract-min = min + delete

→ složitost operací v nejhorším případě je $O(\lg \lg u)$!
= závisí na velikosti množiny klíčů

n	$\lg n$	$\lg \lg n$
2^{32}	32	5
2^{64}	64	6
2^{128}	128	7

= pro řídké u je to v konstantním čase.

→ paměťová složitost: $O(u)$ ⇒ nedá se použít pro velké u.

ZÁKLAD: PŘÍMÉ ADRESOVÁNÍ

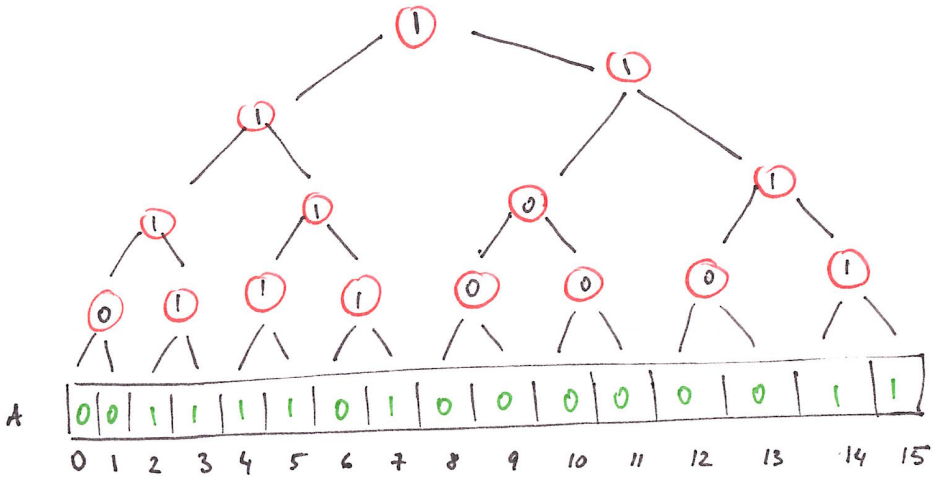
→ pole s u prvky, klíč = index

→ políčko v poli ≡ 1 bit ≡
$A[x] = \begin{cases} 0 & A \text{ neobsahuje } x \\ 1 & A \text{ obsahuje } x \end{cases}$

→ INSERT / DELETE / MEMBER v konstantním čase

→ MIN, MAX, SUCCESSOR, PREDECESSOR
$\Theta(u)$: např. známe $A[0]=1$ a $A[u-1]=1$, zbytek 0. SUCCESSOR(0) projde celé pole A.

ZRYCHLENÍ: POSTAVÍME NAD POLEM BINÁRNÍ STROM



→ v uzlech 1 bit, značí: v něm podstromu je alespoň 1 prvek (= jedno políčko je nastaveno na 1)

Máme tedy: v rodiči je 1, pokud v alespoň jednom potomku je 1.

→ výška stromu je logaritmická
(lze si ji představit i jako bin. vyh. strom, klíče v listech jsou indexy, klíče v červ. uzlech jsou 0.5, 1.5, ... 4.5)

→ MINIMUM → jdeme od kořene do listu, pořád zleva do nejlevějšího potomka, který obsahuje 1.
(pokud neexistuje, pak je strom prázdný: to zjistíme už v kořeni)

→ MAXIMUM JE ANALOGICKÉ.

→ SUCCESSOR(x)

začneme v listu s indexem x. Jdeme směrem ke kořeni, zastavíme v momentě, kdy jsme do uzlu přišli zleva a současně má tento uzel pravého potomka s hodnotou 1.

Označme takový uzel z.

K uzlu z najdeme minimum v jeho podstromu (= jdeme nejlevější cestou obsahující 1 do listu).

→ PREDECESSOR je analogický

→ INSERT(x): vložíme do A[x] hodnotu 1. Potom na cestě do kořene dáme všem uzlům hodnotu 1.

→ DELETE(x):

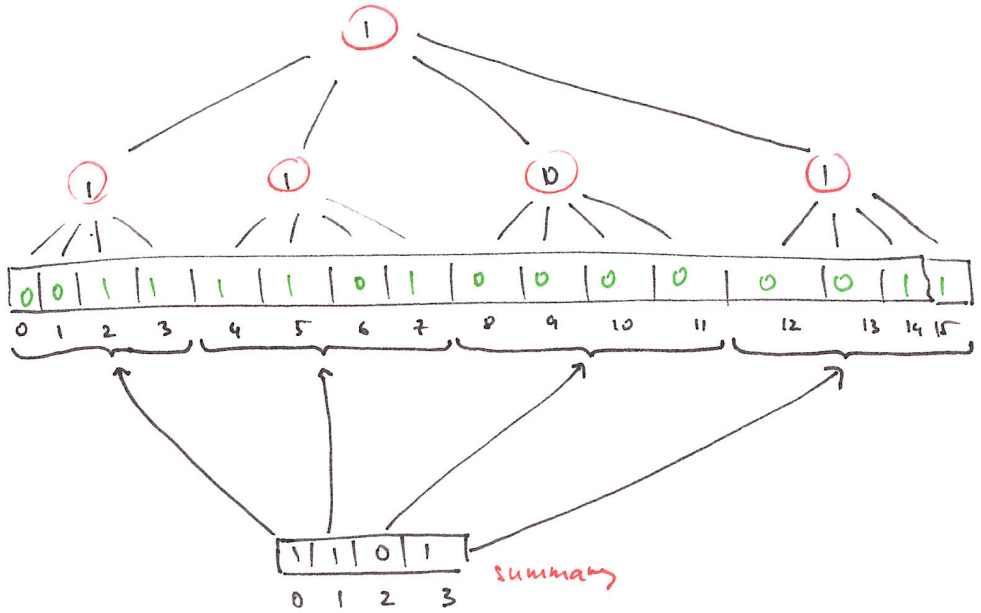
nastavíme A[x] na 0

přepočítáme hodnoty všech uzlů na cestě do kořene:

→ složitosti operací jsou $\Theta(\lg u)$.

POSTAVÍME NAD POLEM VĚTRÁNÍ STROM

$u = 2^{2k}$ ... $\sqrt{u}$ je celé číslo (2^k)



→ konstantní výška: 2

→ máme $\sqrt{u}$ shluků bitů, každý dlouhý $\sqrt{u}$ bitů

potomky kořene máme v poli summary s $\sqrt{u}$ prvky.

→ A[x] je ve shluku $\lfloor x/\sqrt{u} \rfloor$, shluk i je $A[i\cdot\sqrt{u}] \dots A[(i+1)\sqrt{u}-1]$

→ INSERT: summary $[\lfloor x/\sqrt{u} \rfloor] \leftarrow 1$, $A[x] \leftarrow 1$ } $\Theta(1)$

→ MIN: 1) nejlevější 1 v summary → index shluku
2) v daném shluku najdeme nejlevější 1
} $\rightarrow \Theta(\sqrt{u})$

→ SUCCESSOR(x) doprava

1) nejdříve jdeme v rámci shluku ~~doleva~~ od pozice x.
Pokud najdeme 1, ji to výsledek (= index, na kterém tato jednička ji)

2) jinak $i = \lfloor x / \sqrt{u} \rfloor$ a v poli summary najdeme první index $j > i$ tak, že summary[j] = 1. (pokud neexistuje, ji x max. prvek).

3) Najdeme index nejlevější jedničky ve ~~této~~ shluku j, a ten vrátíme jako výsledek.

SLOŽITOST: $O(\sqrt{u})$

→ DELETE(x)

1) A[x] ← 0
2) summary[$\lfloor x / \sqrt{u} \rfloor$] ← logický součet bitů ve celém shluku $\lfloor x / \sqrt{u} \rfloor$.

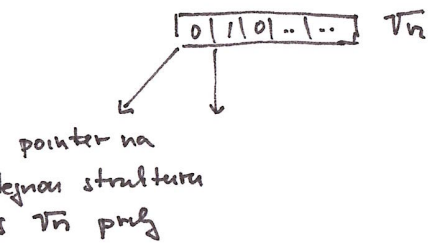
SLOŽITOST: $O(\sqrt{u})$.

Ještě to musíme zlepšit: Dokonce $O(\sqrt{n})$ ji pomalejší než $O(\lg n)$!

Uvidíme, že zrychlení půjde z použití rekurzivní struktury.

REKURZIVNÍ STRUKTURA

REKURZIVNÍ PŘÍPAD



BASE-CASE

pro $u = 2$.

Pro jednoduchost předp. že $u = 2^{(2^k)}$ (toto později odstraníme)

velikosti struktur v jednotlivých patrech rekurzivního stromu jsou

$u, \sqrt{u}, \sqrt[4]{u}, \sqrt[8]{u}, \ldots 2.$

Jak je strom vysoký?

$T(u) = T(\sqrt{u}) + 1$ | $m = \lg u$, $u = 2^m$

$T(2^m) = T(2^{m/2}) + 1$ | přejmenujeme $T(2^m)$ na $S(m)$

$S(m) = S(m/2) + 1$ | Master theorem

$S(m) = O(\lg m)$

$T(2^m) = O(\lg m)$ → $T(n) = O(\lg \lg n)$

Alternativní úvaha

u lze reprezentovat pomocí $\lg u$ bitů

$\sqrt{u}$ lze reprezentovat pomocí $\frac{\lg u}{2}$ bitů

$\sqrt[4]{u}$ — " — $\frac{\lg u}{4}$ bitů

⋮

počet bitů se zmenší na polovinu

z b bitů se na ~~zbitý~~ 1 bit ($u=2$) dostanu pomocí $\lg b$ rozpůlení. Tj pokud $b = \lg u$, potřebujeme $\lg \lg u$ úkonů.

Index rekurzivní struktury, číslo v rekurzivní struktuře

$\lfloor x / \sqrt{u} \rfloor$ = high(x) = horních $(\lg(u))/2$ bitů x ⇐ pozice v summary

$x \bmod \sqrt{u}$ = low(x) = dolních $\lg u / 2$ bitů x ⇐ pozice v rámci ~~bloku~~ rekurzivní struktury.

x = index$(y, z) = y \cdot \sqrt{u} + z$

potom x = index(high(x), low(x)). ⇐ jak dostat číslo zpět

Proto- van Emde Boas stromy

→ přiblížení k opravdovému van Emde Boas stromům (nevadí $O(\lg \lg n)$, ale uvidíme proč a potom to opravíme).

→ uvidíme ale podstatu rekurzivní struktury

→ poté předpokládáme $u = 2^{(2^k)}$

→ struktura (proto-vEB)

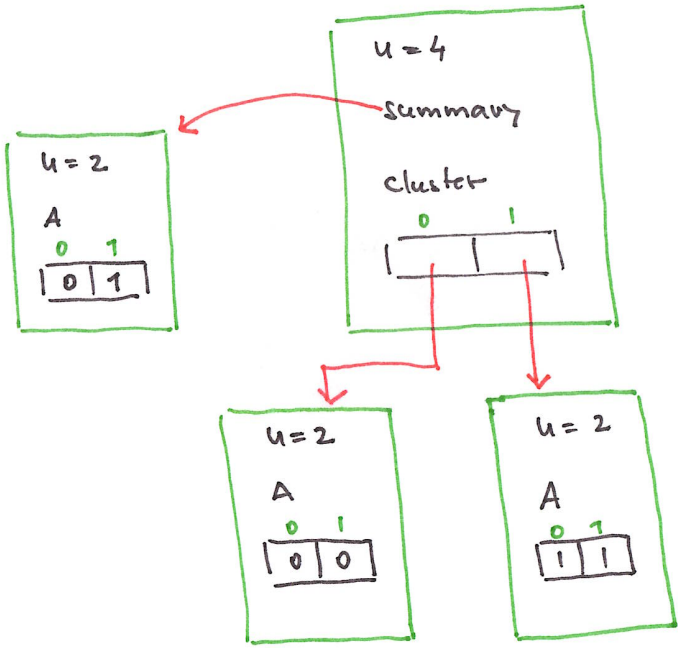
```
struct {
    u,   // velikost struktury, klíče 0, ... u-1.
    summary, // pointer na proto-vEB
             strukturu s √u prvky.
             Tato struktura obsahuje
             indexy rekurzivních proto-vEB
             struktur (obsahujících reálné
             klíče), které nejsou prázdné
    cluster, // pole √u pointerů na proto-vEB
             struktury s √u prvky,
             které slouží k uložení akt.
             klíčů
}
```

→ prvek x $(0 \le x < u)$ je uložen ve struktuře cluster[high(x)] jako prvek low(x)

→ v tomto případě potom summary obsahuje prvek high(x), protože struktura cluster[high(x)] není prázdná.

→ konec rekurze: pokud $u = 2$, máme ve struktuře bitové pole A kde obsah ukládáme přímo.

Př: $u = 4$ $(2^{(2^1)})$, množina $\{2,3\}$



pro 2 máme high(2) = 1
low(2) = 0

OPERACE

MEMBER (V, x)

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

Složitost:

$$T(n) = T(\sqrt{n}) + O(1) \quad \rightsquigarrow \quad O(\lg \lg n)$$

MIN (V)

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

base case: vyhledáváme hrubou silou
NIL značí neexistenci minima

najdeme v podstromech s nejmenším indexem

v je prázdné

Složitost MIN

→ Máme 2 rekurzivní zavolání, jedno pro summary a jedno v podstruktuře

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

$T(2^m) = 2T(2^{m/2}) + O(1)$ // přejmenujeme $T(2^m)$ na $S(m)$

$S(m) = 2T(m/2) + O(1)$ // master Thm: $\Theta(m)$

$T(2^m) = \Theta(m)$

$T(u) = \Theta(\lg u)$ ← toto není $O(\lg \lg u)$

problém je, že máme o 1 rekurzivní zavolání víc

Složitost SUCCESSOR

→ 2 rekurzivní volání + volání MIN

$T(u) = 2T(\sqrt{u}) + \Theta(\lg u)$

Řešení je $\Theta(\lg u \cdot \lg \lg u)$

Problémy: moc rek. zavolání
neefektivní min.

```
SUCCESSOR (V, x)
if  V.u == 2
    if x == 0 && V.A[1] == 1  return 1     } base case:
    else return NIL                        } zkontrolujeme hrubou silou
.
.
.
else
    offset ← SUCCESSOR (V.cluster[high(x)], low(x))     // hledáme ve stejné podstruktuře jako x
    if offset ≠ NIL return index(high(x), offset)

    succ-cluster ← SUCCESSOR (V.summary, high(x))       // najdeme nejmenší větší neprázdnou podstrukturu
    if succ-cluster == NIL  return NIL                  // pokud existuje, vrátíme její minimum
    else
        offset ← MINIMUM (V.cluster[succ-cluster])
        return index (succ-cluster, offset)
```

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

2 rek. zavolání, složitost jako u MIN, $O(\lg u)$

DELETE

→ komplikovaná úprava V.summary, musíme projít celou podstrukturu, jelikož není prázdná
(alternativně můžeme počítat počet prvků v atributu).
↳ to je stejně $\Theta(\lg n)$, protože máme 2 rek. zavolání.

Opravdový van Emde Boas strom

→ 1) u nemusí být $u = 2^{(2^k)}$ pro nějaké $k \geq 1$.
Stačí $u = 2^k$ pro $k \geq 1$.

Pokud $u = 2^{2k+1}$ (pro $k \geq 1$), pak rozdělíme $\lg u$ bitů na

high(x) ... horních $\lceil \lg u / 2 \rceil$ bitů
low(x) ... dolních $\lfloor \lg u / 2 \rfloor$ bitů

definujeme tedy:

$$\uparrow\sqrt{u} = 2^{\lceil \lg u/2 \rceil}$$
$$\downarrow\sqrt{u} = 2^{\lfloor \lg u/2 \rfloor}$$

Např.
$u = 2^3 = 8$
$\uparrow\sqrt{u} = 4 = 2^{\lceil 3/2 \rceil}$
$\downarrow\sqrt{u} = 2 = 2^{\lfloor 3/2 \rfloor}$

Vždycky platí $u = \uparrow\sqrt{u} \cdot \downarrow\sqrt{u}$ a
pokud $u = 2^{(2^t)}$ pro $t \geq 1$, $\uparrow\sqrt{u} = \downarrow\sqrt{u}$.

S použitím nové notace máme

$high(x) = \lfloor x / \downarrow\sqrt{u} \rfloor$
$low(x) = x \bmod \downarrow\sqrt{u}$
$index(y,z) = y \cdot \downarrow\sqrt{u} + z$

→ 2) upravíme strukturu pro uzel

(vEB)

```
struct {
    u,        // stejné jako u proto-vEB
    min,      // min. a max. prvek,
    max,      // vše uloženo
    summary,  // vEB pro ↑√u prvků
    cluster,  // pole o velikosti ↑√u
              // pro ukazatele na vEB pro ↓√u prvků
}
```

Fungování položek min a max

→ vEB neobsahuje žádný prvek: min = max = NIL

→ pokud min ≠ NIL, potom min není obsažen v žádné ze struktur odkazovaných v položce cluster

→ pokud vEB obsahuje 1 prvek, potom min = max

→ pokud vEB obsahuje 2 nebo více prvků, potom min ≠ max. a Prvek max je obsažen ve správné struktuře odkazované v položce cluster.

→ pokud $|u| == 2$, nepoužíváme pole A, ale prvky uložíme jinou pomocí min a max

$\emptyset$ ... min = max = NIL
$\{1\}$ ... min = max = 1
$\{0\}$ ... min = max = 0
$\{0,1\}$ ... min = 0, max = 1

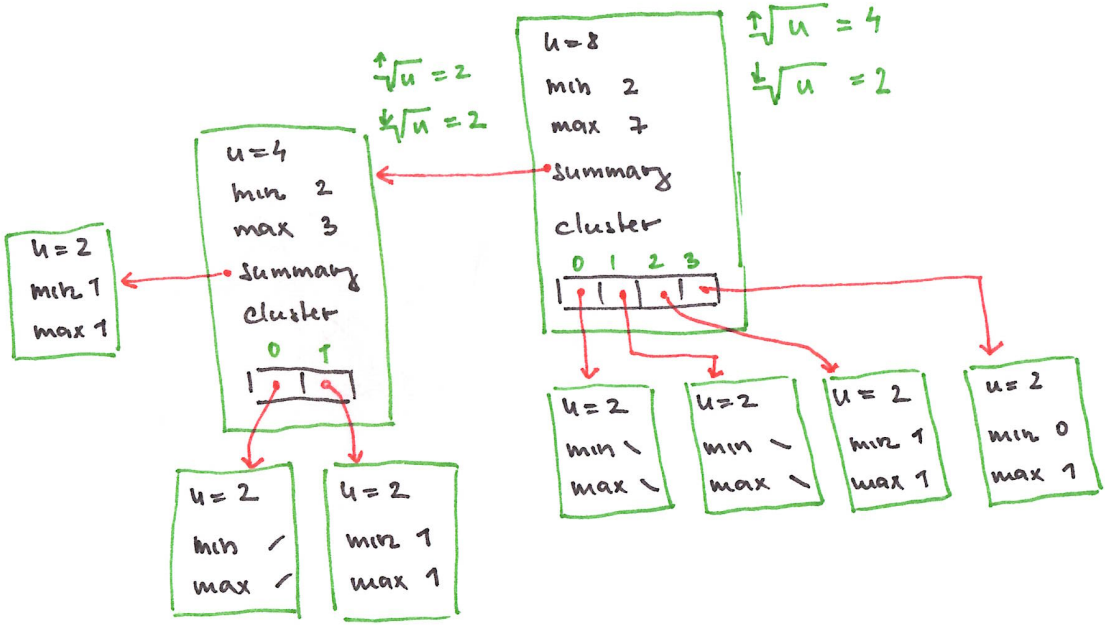
Co nám potřeby min a max dají?

→ procedury MIN, MAX v konstantním čase

→ v proc. SUCCESSOR nemusíme kontrolovat, jestli
* následník x leží v cluster[high(x)], ~~pro~~ rekurzivním zavoláním, protože to ví
právě když low(x) < cluster[high(x)].max !

→ vkládání do prázdného a mazání z
jednoprvkového stromu lze provést v
konstantním čase.

Př: $u = 8$, $\{2, 5, 6, 7\}$



OPERACE

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))
```

Složitost: $\downarrow\sqrt{u} \leq \sqrt{u}$, proto zůstává jako u proto vEB, $\Theta(\lg \lg u)$.

~~FR~~ SUCCESSOR (V, x)

```
BASE-CASE:
if V.u == 2
    if x == 0 && V.max == 1   return 1
    else return NIL
successor je ujmenší prvek:
else if V.min ≠ NIL && x < V.min
    return V.min.
else
    m-low ← V.cluster[high(x)].max
successor je v podstromu, ve kterém je i x:
    if m-low ≠ NIL && low(x) < ~~max~~ m-low
        offset ← SUCCESSOR(V.cluster[high(x)], low(x))
        return index(high(x), offset)
najdu podstrom, ve kterém je successor:
else
    succ-cluster ← SUCCESSOR(V.summary, high(x))
    if succ-cluster == NIL
        Return NIL
    else
        offset ← V.cluster[succ-cluster].min
        return index(succ-cluster, offset)
```

Složitost operace successor

→ v nejhorším případě (při hledání succ-cluster) máme jedno rek. zavolání pro podstrom velikosti $\sqrt[\uparrow]{u}$

$T(u) = T(\sqrt[\uparrow]{u}) + O(1)$ // $m = \lg u$

$T(2^m) = T(2^{\lceil m/2 \rceil}) + O(1)$ // $\lceil m/2 \rceil \le 2m/3$, pro $m \ge 2$

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

$S(m) \le S(2m/3) + O(1)$ // master theorem → $\Theta(\lg m)$

$T(2^m) \le \Theta(\lg m)$

$T(u) = \Theta(\lg \lg u)$ ← to chceme!

INSERT (V, x)

```
if V.min == NIL                     } = EMPTY-INSERT (V,x)
    V.min ← x                           V.min ← x
    V.max ← x                           V.max ← x
else
    if x < V.min
        x ↔ V.min
    if V.u > 2
        if V.cluster[high(x)].min == NIL        } do prázdného podstromu
            INSERT (V.summary, high(x))
            EMPTY-INSERT (V.cluster[high(x)], low(x))
        else                                    } do plného stromu
            INSERT (V.cluster[high(x)], low(x))
    if x > V.max                                } uložení správného max
        V.max ← x
```

Složitost INSERT

→ vždycky máme 1 rek. zavolání
- vložení do summary, nebo
- vložení do podstromu, který není prázdný

→ $T(u) = T(\sqrt[\uparrow]{u}) + \Theta(1)$

tj. $O(\lg \lg u)$ ← OK.

DELETE (V, x)   (~~před~~ předp. x je ve V)

```
if V.min == V.max                       } mažu poslední prvek
    V.min ← V.max ← NIL
else if V.u == 2                        } mažu z base case stromu
    if x == 0   V.min ← 1
    else V.min ← 0
    V.max ← V.min
else
    if x == V.min                       // musím najít nový min a ten potom smazat
        f ← ~~min~~ V.summary.min
        x-low ← V.cluster[f].min
        x ← index(f, x-low)
        V.min ← x                       ← dotedy je v x prvek ke smazání
    ~~if V.cluster[high(x)].min == NIL~~
    DELETE (V.cluster[high(x)], low(x))  // mažu x
    if V.cluster[high(x)].min == NIL
        DELETE (V.summary, high(x))
        if x == V.max                   // mažu maximum?
            sm ← V.summary.max
            if sm == NIL   V.max ← V.min
            else V.max ← index(sm, V.cluster[sm].max)
    else if x == V.max                  // mažu maximum?
        V.max ← index(high(x), V.cluster[high(x)].max)
```

(poznámky na okraji:) nové maximum je ze stejného podstromu; nové maximum je z dalšího podstromu.

Složitost DELETE

→ ze dvou rekurzivních zavolání je vždy jedno v konstantním čase:

→ buď mažeme poslední prvek z V.cluster[high(x)] (v konstantním čase), a potom v tet mažeme z high(x) ze V.summary

→ nebo mažeme z V.cluster[high(x)] prvek, který není poslední a k mazání z V.summary vůbec nedojde.

To vede na rekurenci

$$T(u) \leq 2T(\sqrt[2]{u}) + O(1),$$

tedy složitost je $\Theta(\lg \lg n)$