

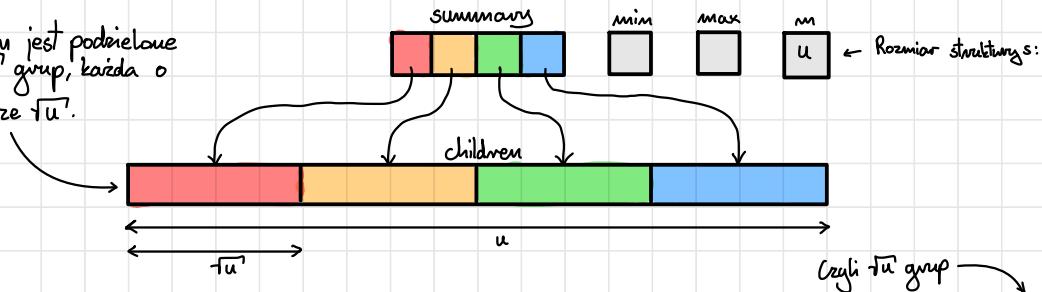
vom Ende Boas,

Drzewo van Emde Boasa

- Operacje na uniwersum liczb U . Zauważaj to uniwersum będącym traktując jako np. uniwersum liczb 64-bitowych. $u = \sqrt{U}$
- Czasołaci czasowe wszystkich operacji będą uzależnione od u i będą wynosić $\log \log u$. Dla liczb 64-bitowych to jest zaledwie 6 operacji razy jaką stała c .
- Operacje: insert, successor, predecessor, delete, find, min, max

Wyobraźmy sobie następującą strukturę „s”:

children jest podzielone na $\lceil u \rceil$ grup, każda o rozmiarze $\lceil u \rceil$.



Gdzie tablica summary jest rozmiaru $\lceil u \rceil$, a children ma rozmiar u

Tablica summary przechowujeć będzie tylko bity w której grupie znajduje się jakiś element.

Innymi słowy, jeśli dany element znajduje się w pierwszej grupie, to przechowujemy zapalony bit w pierwszej pozycji

Zanim przejdziemy dalej zobaczymy jak możemy dodawać elementy.

Aby dodać liczbę $x = 01001011$, dzielimy ją na pół

$$\left\{ \begin{array}{l} \left\lfloor \frac{x}{2^k} \right\rfloor \quad x \bmod 2^k \\ \downarrow \qquad \qquad \qquad \downarrow \\ \text{high}(x) \quad \text{low}(x) \end{array} \right. \quad \text{Num}$$

W strukturze s przekształcamy bit pod indeksem $\text{low}(x)$ w grupie $\text{high}(x)$.

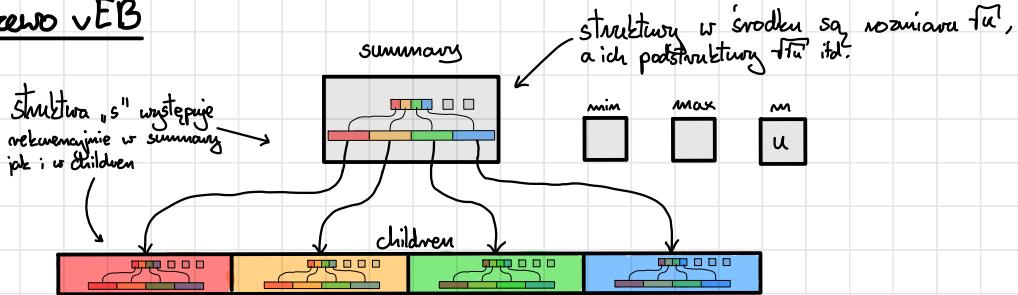
Jeśli jednak x jest mniejszy niż minimalna wartość dla danego poddrzewa lub ta wartość nie została jeszcze ustalona - ustawiamy x w miejsce min, a min wstawiamy do children.

Struktura "s" jednak nie daje nam możliwości zakładania na pożądane. To dlatego, że drzewo van Emde Boasa jest dodatkowo strukturą rekurencyjną.

Wyobraźmy sobie strukturę "s", która jest budowana rekurencyjnie. Każda grupa to osobna struktura "s" oraz summary jest również strukturą "s". Taką strukturę nazywamy drzewem van Emde Boasa (vEB).

Dla każdej mniejszej struktury "s" będziemy przypisywać jej wartość w dwa razy mniejszą niż rodzic, zatem rozmiar children będzie \sqrt{m} .

Drzewo vEB



Zauważ, że wartość minimalna przechowywana jest tylko w tym poddrzewie - nigdzie indziej. Zatem minimum przechowywane w konczeniu jest minimum dla całej struktury.

Analogicznie jest z max, jednakże max może występować dodatkowo jako wartość głębszej w strukturze.

Wstawianie

Zdefiniujmy $\text{simple_insert}(x, S)$ jako

Simple_insert (x, S)

$$\forall v. \text{min} = x$$

$$\forall v. \text{max} = x$$

Simple insert ma złożoność $O(1)$

Na wstawianie elementu do vEB rozważmy 4 przypadki:

- drzewo S jest puste

$\text{simple_insert}(x, S)$

- x jest mniejszy niż $S.\min$ tj. $x < S.\min$

$\text{swap}(x, S.\min)$

$\text{insert}(x, S)$ ← Rekurencyjny insert

- Jeśli w grupie do której chce my dodać x nie mieści się nic mniej niż x , to

$\text{simple_insert}(\text{low}(x), S.\text{children}[\text{high}(x)])$

→ $\text{insert}(\text{high}(x), S.\text{summary})$

- Jeśli coś jest w grupie, do której wstawiamy wartość x

$\text{insert}(\text{low}(x), S.\text{children}[\text{high}(x)])$

Na koncu aktualizujemy $S.\max$, jeśli $x > S.\max$

Każdy insert aktualizujemy tylko raz rekurencyjnie na vEB normowanym \sqrt{u} , zatem:

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

Peten kod wstawiania zaprezentowany ma wykładezie:

Insert(x, S)

Jesli drzewo jest puste

if $S.\text{min} == \text{null}$

$S.\text{min} = x$

return

Jesli x jest mniejszy od min, to podmieniamy
; wstawianie teraz min do drzewa

if $x < S.\text{min}$

swap(x, S.min)

if $S.\text{children}[\text{high}(x)]$ jest pusta

insert($\text{high}(x)$, S.summary)

$s.\text{children}[\text{high}(x)].\text{min} = \text{low}(x)$

else

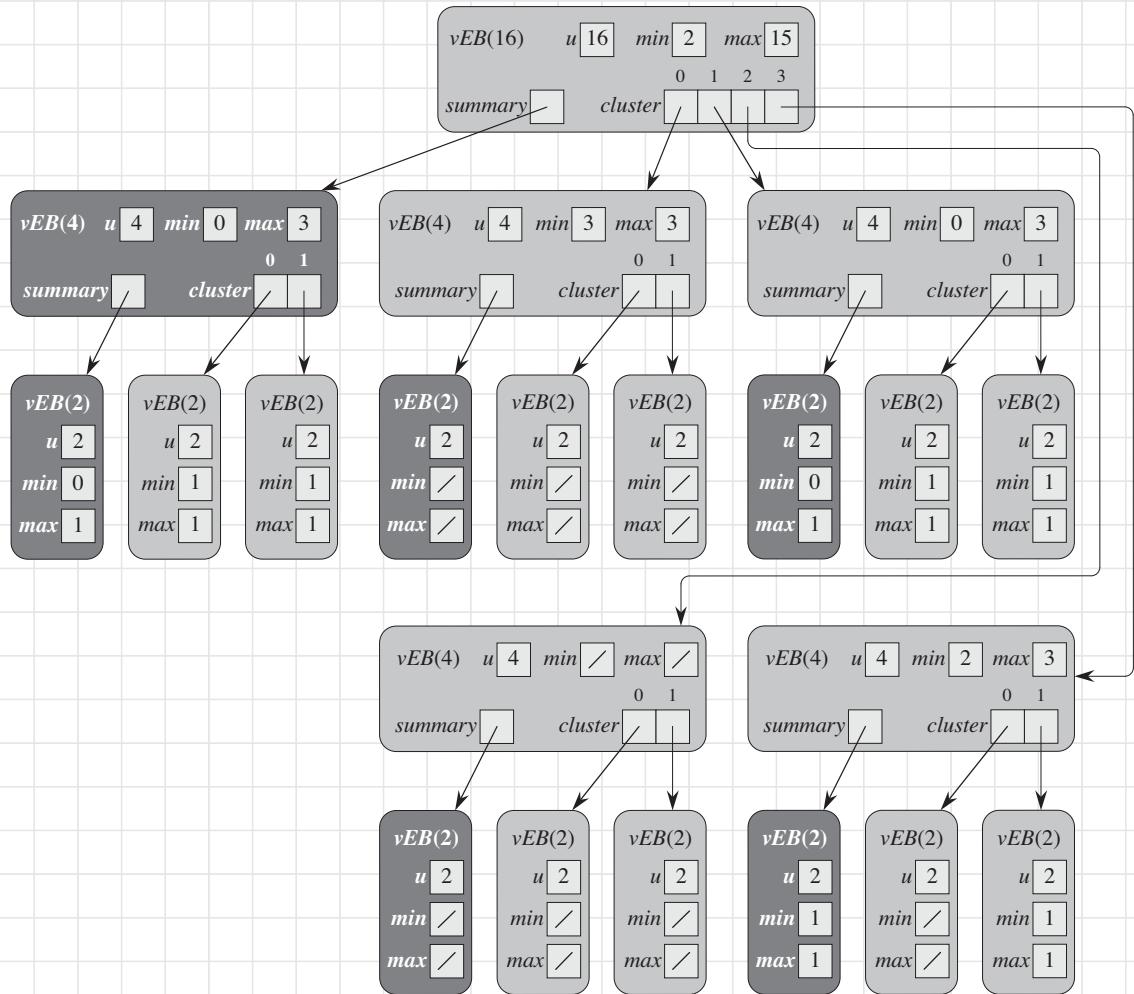
insert($\text{low}(x)$, $s.\text{children}[\text{high}(x)]$)

if $x > S.\text{max}$

$S.\text{max} = x$

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

Dla Tatujskiego zrozumienia przedstawiam tutaj drzewo vEB, $u=16$, do którego umieszczone elementy: 2, 3, 4, 5, 7, 14, 15.



Źródło: Introduction to Algorithms : Third Edition

Successor(x, S)

Czy następnik znajduje się w tej grupie?

if $\text{low}(x) < S.\text{children}[\text{high}(x)].\text{max}$

$j = \text{successor}(\text{low}(x), S.\text{children}[\text{high}(x)])$

return $\text{high}(x) \cdot \lceil \frac{S}{m} \rceil + j$

else

↑ tą wartością przedstawiamy w strukturze.
Jest to $\lceil \frac{S}{m} \rceil$.

$i = \text{successor}(\text{high}(x), S.\text{summary})$

return $S.\text{children}[i].\text{min} + i \cdot \lceil \frac{S}{m} \rceil$

Jako, że przedstawiamy
tylko $\text{low}(x)$ w strukturze,
to wykonujemy odzysk

Złożoność czasowa:

$$T(u) = O(\lceil \frac{u}{m} \rceil) + O(1) \rightarrow O(\log \log u)$$

Na usuwanie z vEB warzącym 3 przypadki

- Jeśli usuwamy minimum
weź z najbliższego klastru minimum;
zamień z wartością usuwaną x .

$\text{Delete}(\text{S}. \text{children}[\text{high}(x)], \text{low}(x))$ ← Wykona się tylko raz
gdy to był ostatni element

- Jeśli grupa jest pusta, to usuń z podsumowania
 $\text{Delete}(\text{S}. \text{summary}, \text{high}(x))$

- Jeśli to był max
weź z najbliższego klastru maximum;
ustaw jako max

Delete(x, S)

$\text{if } x == S.\text{min}$

$i = S.\text{summary}.\text{min}$

$x = i \cdot \sqrt{|S|} + S.\text{children}[i].\text{min}$

$S.\text{min} = x$ ← None minimum to następcą wartości min

$\text{delete}(S.\text{children}[\text{high}(x)], \text{low}(x))$ ← Jeżeli ten if się wykonie, to ten delete wykonanie O(1), bo to ta grupa z jednym elementem

$\text{if } S.\text{children}[\text{high}(x)].\text{min} \text{ jest pusta}$ ← jeśli to był ostatni element w tej grupie

$\text{delete}(S.\text{summary}, \text{high}(x))$ ← to wtedy usunął tę grupę z summary

$\text{if } x == S.\text{max}$

$i = S.\text{summary}.\text{max}$ ← Ostatni nie pusty klaster

$S.\text{max} = i \cdot \sqrt{|S|} + S.\text{children}[i].\text{max}$ ← wieź jego max

Złożoność czasowa:

$$T(u) = T(\frac{u}{2}) + O(1) \rightarrow O(\log \log u)$$