2.4 Drzewa van Emde Boasa

- Próbujemy utworzyć strukturę danych na której możemy wykonać operację: Insert, Delete oraz Successor.
- Znamy już taką strukturę danych! To zbalansowane drzewa poszukiwań binarnych. Działają w czasie $O(\log n)$.
- Chcemy pobić tą złożoność przez dodanie dodatkowego założenia, że klucze, które wstawiamy to liczby z przedziału $0..u-1$.
- Pierwszy krok. Wyobraźmy sobie, że chcemy tylko wstawiać i usuwać elementy. Potrafimy to zrobić w czasie stałym na tablicy!
- Operacja następnika może działać w czasie liniowym. Spróbujmy ją usprawnić!
- Dzielimy tą tablicę na $\sqrt{n}$ równych części i dodajemy do każdej z nich informację czy jest niezerowa.
- Wstawianie w czasie stałym, successor w czasie pierwiastkowym, usuwanie w czasie pierwiastkowym (usuwanie troche bruździ - usuńmy go na razie; pun intented).
- Rozpiszmy sobie operacje jakie wykonujemy na tablicach (pseudokod)
- Zobaczmy, że szukanie "następnika" w danej tablicy trochę czasu zajmuje - jak moglibyśmy to przyśpieszyć... hmmmm...
- Terminologia
$$x = i\sqrt{u} + j$$
- przy czym $0 \leqslant j < \sqrt{u}$.
- $low(x) = j = x \mod \sqrt{u}$
- $high(x) = i = \lfloor x/sqrtu \rfloor$
- $index(i, j) = i\sqrt{u} + j = x$
- W komputerach się to implementuje inaczej, bo ma to fajne uzasadnienie binarne.

- Struktura drzew vEB(U) składa się z:
  - tablica drzew vEB($\sqrt{U}$) V.cluster[$\sqrt{U}$]
  - drzewo vEB($\sqrt{U}$) V.summary

```
Insert(V, x)
  Insert(V.cluster[high(x)], low(x))
  Insert(V.summary, high(x))

Successor(V, x)
  i = high(x)
  j = successor(V.cluster[i], low(x))
  if (j == oo)
    i = successor(V.summary, high(x))
    j = successor(V.cluster[i], -1)
 return index(i, j)
```

- Złożoność Inserta: $T(u) = 2T(\sqrt{u}) + O(1) = O(\log u)$.
- Złożoność Succ: $T(u) = 3T(\sqrt{u}) + O(1) = O(\log^{\log 3} u)$.
- Naprawmy successor.
- Jak pozbyć się succesora od -1?
- Dla każdego drzewa van Emde Boasa pamiętajmy minimum!

```
Insert(V, x)
  if x < V.minimum
    V.minimum = x
  Insert(V.cluster[high(x)], low(x))
  Insert(V.summary, high(x))

Successor(V, x)
  i = high(x)
  j = successor(V.cluster[i], low(x))
  if (j == oo)
    i = successor(V.summary, high(x))
    j = V.cluster[i].minimum
 return index(i, j)
```

- Dalej poprawiamy Succesor.
- Gdybyśmy tylko wiedzieli wcześniej czy pierwsze wywołanie successora się powiedzie czy nie.
- Rozwiązanie: pamiętajmy dla każdego drzewa vEB maximum!

```
Insert(V, x)
  if x < V.minimum
    V.minimum = x
  if x > V.maximum
    V.maximum = x
  Insert(V.cluster[high(x)], low(x))
  Insert(V.summary, high(x))

Successor(V, x)
  i = high(x)
  if low(x) < V.cluster[i].maximum
    j = successor(V.cluster[i], low(x))
  else
    i = successor(V.summary, high(x))
    j = V.cluster[i].minimum
 return index(i, j)
```

- Jak naprawić insert?
- Możemy poprawić nieco, niewstawiając do summary jeśli nie trzeba.
- Teraz wstawiamy podwójnie tylko jeśli cluster jest pusty.
- Propagowanie leniwe działa! Ale zróbmy prościej.
- Skoro wstawiamy nowy element do pustego clustra, to zapiszemy go do minimum! Po co zapisywać go w tablicy, skoro mamy zapamiętanego go w minimum?

```
Insert(V, x)
  if V.minimum == None
    V.minimum = x
    V.maximum = x
```

```
  if x < V.minimum
    V.minimum <-> x
  if x > V.maximum
    V.maximum = x
  if V.cluster[high(x).minimum == None
    Insert(V.summary, high(x))
  Insert(V.cluster[high(x)], low(x))
```

```
Successor(V, x)
  if x < V.minimum
    return V.minimum
  i = high(x)
  if low(x) < V.cluster[i].maximum
    j = successor(V.cluster[i], low(x))
  else
    i = successor(V.summary, high(x))
    j = V.cluster[i].minimum
 return index(i, j)
```

Usuwanie to odwrócenie Inserta.

```
Delete(V, x)
  if x  = V.minimum
    i = summary.minimum
    if i == None
      V.min = V.max = None
      return
    x = V.min = index(i, V.cluster[i].min)
  Delete(V.cluster[high(x)], low(x))
  if V.cluster[high(x).minimum == None
    Delete(V.summary, high(x))
  if x == V.maximum
    if V.summary.maximum = None
      V.maximum = V.minimum
    else
      i = V.summary.maximum
      V.maximum = index(i, V.cluster[i].maximum)
```

- Można udowodnić, że nie da się lepiej niż $O(\log \log u)$.