Drzewa van Emde Boasa

~kolejka

- w ogólnym przypadku: deleteMin $O(\log n)$
- tutaj ograniczamy się do universum $U = \underline{u}$ $\underline{u} = \{0, \ldots, u-1\}$
- i pozwalamy na operacje na kluczach (inne niż porównania)

Cel: deleteMin $O(\log \log n)$

- może dziel i zwyciężaj?

$T(n) = T(\_) + \_$ ← co możemy wstawić, żeby

$T(n) = \log \log n$

$T(n) = T(\sqrt{n}) + \Theta(1)$

Operacje: insert ← zrobić samodzielnie

delete ← skupimy się na tym

succ

pred ← symetryczne

przyjmuje także argumenty spoza obecnego zbioru

Naiwnie

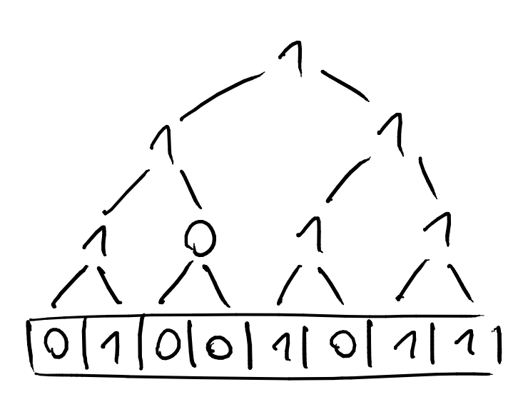
- zbalansowane BST – $O(\log n)$
  - succ/pred $O(1)$ na przeplecionej liście
- tablica następników X
- wektor charakterystyczny

insert $O(1)$

succ $O(n)$

|0|1|0|0|1|0|1|1|

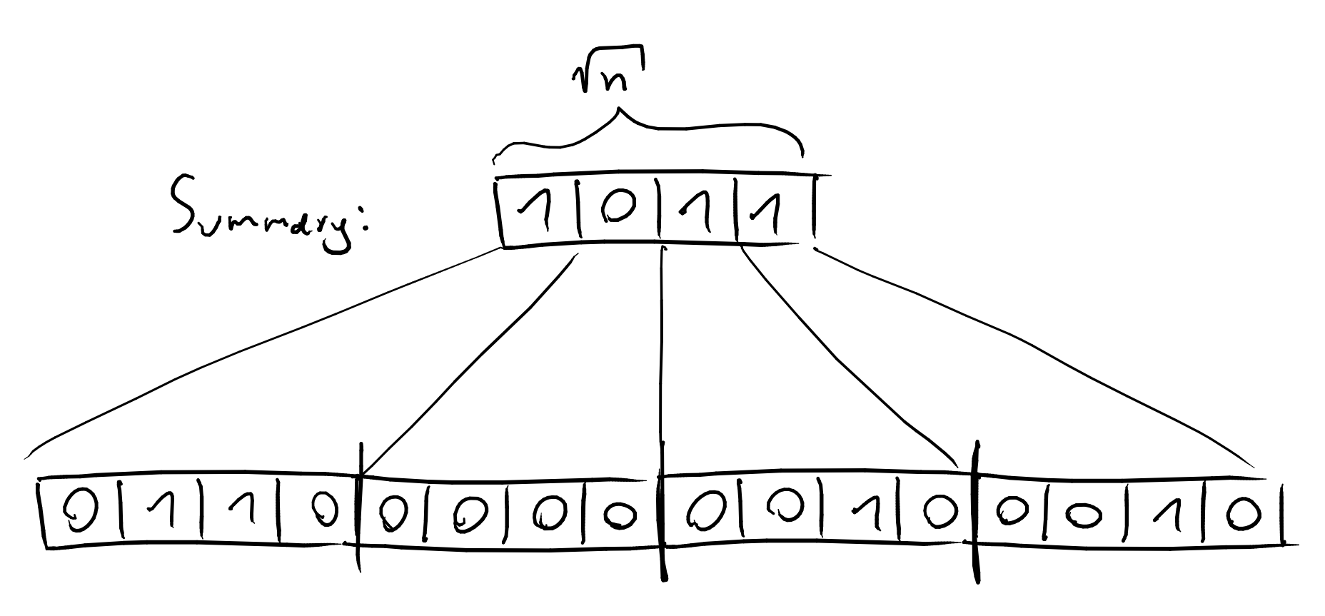
1) Przyspieszmy succ: zbudujmy nad wektorem drzewo binarne



insert: $O(\log n)$

succ: $O(\log n)$

2) Struktura pierwiastkowa

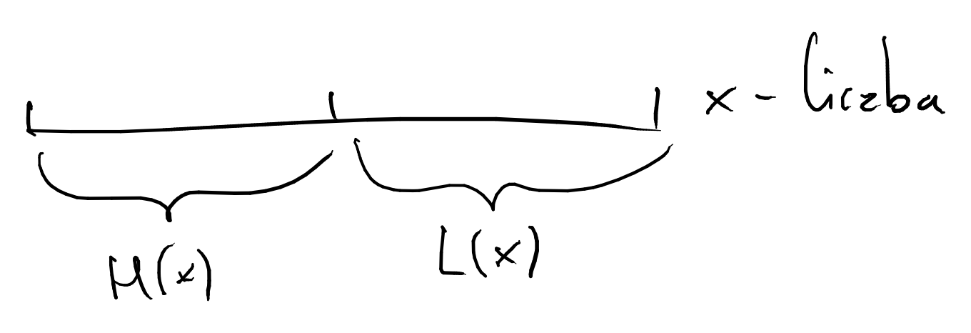


succ: $O(\sqrt{n})$

- niby źle, ale dobrze!

Sprowadziliśmy problem szukania następnika w strukturze n-elementowej do szukania w strukturze $\sqrt{n}$-elementowej!

Oznaczenia:



$x = (01110011)_2$

$L(x) = (0011)_2 = 3$

$H(x) = (0111)_2 = 7$

Indeksowanie:

n=16

nr gr indeks w grupie

```
0000
0001
0010
0011
----
0100
```

Struktura rekurencyjna:

```
S n = {
    sub :: Array (S √n) √n
    summary: S √n
}
```

```
insert x S :=
    insert ( L(x), S.sub[H(x)] )
    insert ( H(x), S.summary )  jeśli S.sub[H(x)] była pusta
                                         ↑
                                   optymalizacja
```

```
succ (x, S) :=
    j ← succ (L(x), S.sub[s][H(x)]
    Some ( H(x) · √|S| + j )
    <|>
     i ← succ (H(x), S.summary)
     j ← succ (-∞, S.sub[i])
     Some  i · √|S| + j
```

Czas: insert : $T(n) = 2T(\sqrt{n}) + \ldots = \Omega(\log n)$

succ : $T(n) = 3T(\sqrt{n}) + \ldots = \Omega(\log^{\log_2 3} n)$

- za dużo wywołań!

Poprawka I: trzymamy min/maks

Struktura rekurencyjna:

```
S n = {
    sub :: Array (S √n) √n
    summary: S √n
    extremums: () + N + N^2
}              ↗     ↑      ↑
          pusta   jeden   min, max
                  element
                  min=max
```

```
succ (x, S) :=
    if   L(x) < S.sub[H(x)].max
    then  j ← succ (L(x), S.sub[H(x)])
              return  H(x) · √|S| + j
    else  i ← succ (H(x), S.summary[x])
              return  S.sub[i].min
```

insert: możemy szybko sprawdzić, czy puste, ale dalej 2 wywołania

Poprawka II: minimum nie trzymamy w tablicy (tylko w dedykowanym polu)

Struktura rekurencyjna:

```
S n = {
    sub :: Array (S √n) √n
    summary: S √n
    extremums: () + N + N^2
}              ↗     ↑      ↑
          pusta   jeden   min, max
                  element
                  min=max
```

```
insert (x, S) :=

    if   x < S.min
    then  insert (S.min, S[min ↦ x])
    else
        when (x > S.max)   S.max := x
        if    S.sub[H(x)] is empty
        then  insert ( H(x), S.summary)
                S.sub[H(x)].min := x

        else  insert (L(x), S.sub[H(x)])
```