

5. (2pkt) Niech  $h(v)$  oznacza odległość wierzchołka  $v$  do najbliższego pustego wskaźnika w poddrzewie o korzeniu  $v$ . Rozważ możliwość wykorzystania drzew binarnych, równoważonych poprzez utrzymywanie następującego warunku:

$$h(\text{lewy syn } v) \geq h(\text{prawy syn } v) \text{ dla każdego wierzchołka } v,$$

do implementacji złączalnych kolejek priorytetowych.

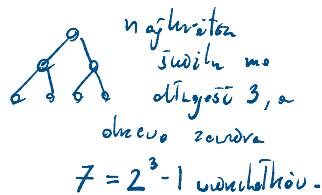
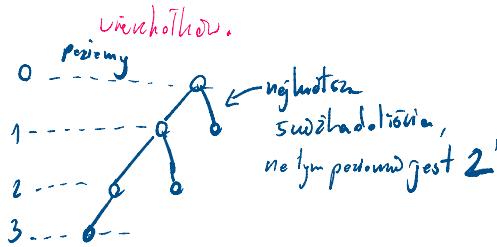
Sprobuj skonstruować kopiec, konstrukcja z dwóch drzew binarnych. Dla kolejnych konstrukcji drzewa lewicowe, tzn. oba we binarne spełniające warunek:

- 1) warunek ogólny, aby oba obiekty są właściwe dla kolejnego wierzchołka
- 2)  $h(v.\text{left}) > h(v.\text{right})$ , gdzie  $v$  to wierzchołek obieku,  $\text{left}$  i  $\text{right}$  to jego podobiekta.

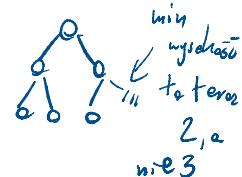
Przez taki obiekt możemy pońieść? Miedzy perymetrem i liczbą obserwacji:

1) Ścieżka przechodząca przez prawe dziecięcia danego wierzchołka najwyżej prowadzi do liścia. Wynika to z prostej faktu, że  $h(v.\text{left}) \geq h(v.\text{right})$ .

2) Jeśli najkrótsza ścieżka od korzenia obieku do liścia wynosi  $k$ , to obieku zawiera co najmniej  $2^k - 1$  wierzchołków. Uwielbiono stąd, że jest to obiekt binarny, zatem na  $i$ -ty poziomie obieku pojawią się  $2^{i-1}$



Jest to najmniejsza możliwa liczba wierzchołków dla tej wysokości minimalnej. Dlatego duchem jednego obiektu o głębokości  $k$  jest  $2^k - 1$  wierzchołków, sprawdzając zmniejszenie maksymalnej wysokości o 1.



Konstrukcja 2 1) i 2) mający umożliwić prosty sposób złączenia powiększeń operacji kopiejkowej do:

```
funkcja scal(HeapA, HeapB)
{
    // przypadki podstawowe
    jeśli HeapA jest pusty -> zwróć HeapB
    jeśli HeapB jest pusty -> zwróć HeapA

    // zachowanie własności kopca
    jeśli wartość korzenia HeapA > niż wartość korzenia HeapB
    {
        zamień miejscami HeapA i HeapB
    }

    Za prawe poddrzewo HeapA ustaw wynik scalenia tego poddrzewa z HeapB (rekurencyjnie)
    // to znaczy HeapA.right = scal(HeapA.right, HeapB)

    // zachowanie warunku, że wysokość lewego poddrzewa jest >= wysokości prawego
    jeśli lewe poddrzewo HeapA jest puste
    {
        zamień miejscami lewe i prawe poddrzewo
        Ustaw długość najkrótszej ścieżki HeapA na 1
    }

    Jeśli wysokość prawego poddrzewa > wysokość lewego poddrzewa HeapA
    {
        Zamień miejscami prawe i lewe poddrzewo
        Zwięksź długość najkrótszej ścieżki HeapA o 1
    }
}
```

Działanie kolejek polega na podpinaniu jednego obieku do prawego ramienia innego obieku w sposób rekurencyjny. Następnie wykorzystywany jest zapisany wraz z obiekiem na obiekcie lewicowym.

Ta procedura jest rekurencyjna, dla wszystkich innych operacji kopiejkowych.

```

    }
    Zwróć HeapA jako kopiec wynikowy
}

```

```

Funkcja usuń-minimum(T)
{
    Zamień drzewo T na drzewo powstałe przez scalenie prawego i lewego poddrzewa T
    // T = merge(T.left, T.right)
    i zwróć nowe drzewo oraz wartość w usuniętym wierzchołku (był minimum, czyli był korzeniem)
    // return T.root, merge(T.left, T.right)
}

```

```

Funkcja dodaj(T, x)
{
    Połącz drzewo T oraz drzewo jednoelementowe, powstałe z elementu x i zwróć wynik
    // return scal(T, tree(x))
}

```

Usunięcie minimum polega na usunięciu żółtego korzenia - minimum i połączaniu jego lewej i prawej poddrzew.

Dodawanie elementu do kolejki polega na traktowaniu nowego elementu jako obrewej jednoelementowej. Utworzyliśmy skonczone obrawe.

## Złożoność

Zauważmy, że wykrojone funkcje opierają się na funkcji  $\text{scal}(A, B)$ , zatem złożoność algorytmu zależy od złożoności tej funkcji. Spróbujmy ją ustalić.

Widzimy, że funkcja  $\text{scal}(A, B)$  wykona się rekurencyjnie na prawym poddrzewie, zatem liczba wykonywanych rekurencji zależy od maksymalnej wysokości tej gałęzi. Wraz z obserwacją 1) i 2), że ta gałąź jest ograniczona w głębokości powinny, że ma wysokość  $h$ . Dodałkowo mamy własność, że liczba wierzchołków obrawe  $n > 2^h - 1$ , zatem

Także mamy południową wariancję:

$$\begin{aligned} n &> 2^h - 1 \\ n+1 &> 2^h / \log_2 \\ \log_2(n+1) &> h \end{aligned}$$

Zauważmy, że w trakcie algorytmu mamy zmianę miejscami kolejki A; B, zatem musimy rozważyć dławione są po maksymum z liczbnych wierzchołków. Zatem złożoność to  $O(\log(\max\{n_1, n_2\}))$ .