Z5

Mamy strukturę drzewa binarnego o własnościach:

- kopiec, czyli wartości lewego dziecka ≥ ojca
- $h(x.left) \geq h(x.right)$, gdzie h to odległość do najbliższego nulla.

Łatwo zauważyć, że idąc zawsze do prawego dziecka najszybciej dotrzemy do nulla.

Co więcej, zauważmy, że jeśli $h(root) = x$ to wszystkie wierzchołki na głębokości $x-1$, mają dwóch synów.

W przeciwnym wypadku $h(root)$ byłoby mniejsze niż x.

A z tego wynika, że rozmiar takiego drzewa to co najmniej $2^x - 1$, a $h(root) \leq \log(n)$, gdzie n to liczba wierzchołków w drzewie.

Skorzystamy z tego faktu implementując funkcje do naszej kolejki.

Będziemy chcieli napisać 3 funkcje: pop_min()
insert (v)
merge(T_1, T_2)

- pop_min()

Aby usunąć element minimalny, wystarczy usunąć korzeń i zmergować synów:

```
def pop_min():
    res = T.val
    T = merge(T.left, T.right)
    return res
```

- insert

Wystarczy utworzyć drzewo składające się tylko z nowej wartości i zmergować z docelowym:

```
def insert(v):
    new_T = T(v)
    merge(T, new_T)
```

- merge

Będziemy rekurencyjnie łączyć kopiec o większym korzeniu do prawego poddrzewa drzewa o mniejszym korzeniu przy okazji dbając o zachowanie obu własności drzewa.

```
def merge(T1, T2):
    if T1 == NULL:
        return T2
    if T2 == NULL:
        return T1

    if T1.value > T2.value:          // chcemy łączyć T2 do poddrzewa T1
        swap(T1, T2)

    T1.right = merge(T1.right, T2)   // wie null, bo T2 != null

    if T1.left == NULL:              // przywracamy niemość leftist tree
        swap(T1.left, T1.right)
        T1.h = 1
    else:
        if T1.right.height > T1.left.height:
            swap(T1.right, T1.left)
        T1.h = T1.right.h + 1        // poza ifem, ale w else

    return T1
```

Złożoność operacji:

- merge - w najgorszym przypadku będziemy iść wzdłuż prawych krawędzi T_1 i T_2 czyli $O(\log_2(|T_1|) + \log_2(|T_2|))$

- pop-min() - analogicznie $O(\log_2(|T_1.left|) + \log_2(|T_1.right|))$

- insert - tutaj mergujemy tylko z 1 wierzchołkiem, więc $O(\log_2(n))$

28)

Rozważmy pewną modyfikację drzewa AVL. W każdym węźle będziemy trzymać dodatkowe 3 wartości:

- mindiff
- wartość skrajnie lewego syna
- wartość skrajnie prawego syna.

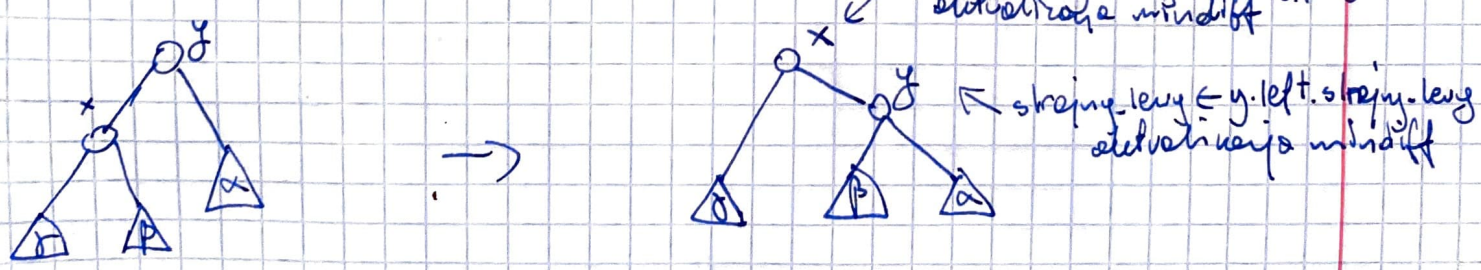
Wartość mindiff dla wierzchołka będziemy wyliczać biorąc min z:

- mindiff lewego syna
- mindiff prawego syna
- wartość - skrajny lewy prawego syna (inorder successor)
- skrajny_prawy - wartość (inorder predecessor)
  lewego syna

Skrajne wartości uzyskujemy od synów wierzchołka.

W trakcie drzewa będziemy mindiff całego miasta.

Jak updatować przy rotacjach:



Po operacjach insert i delete po prostu idziemy w górę drzewa aktualizując wartości. Czas $O(\log n)$

23)

- Zauważmy, że liście zawsze mają balans = 0, więc nie potrzebują trzymać informacji
- W przypadku wierzchołków o tylko 1 synie, to czy jest po prawej czy po lewej stronie jednoznacznie wyznacza balans
- Zatem możemy trzymać informację o wierzchołkach korzystając z ich synów: