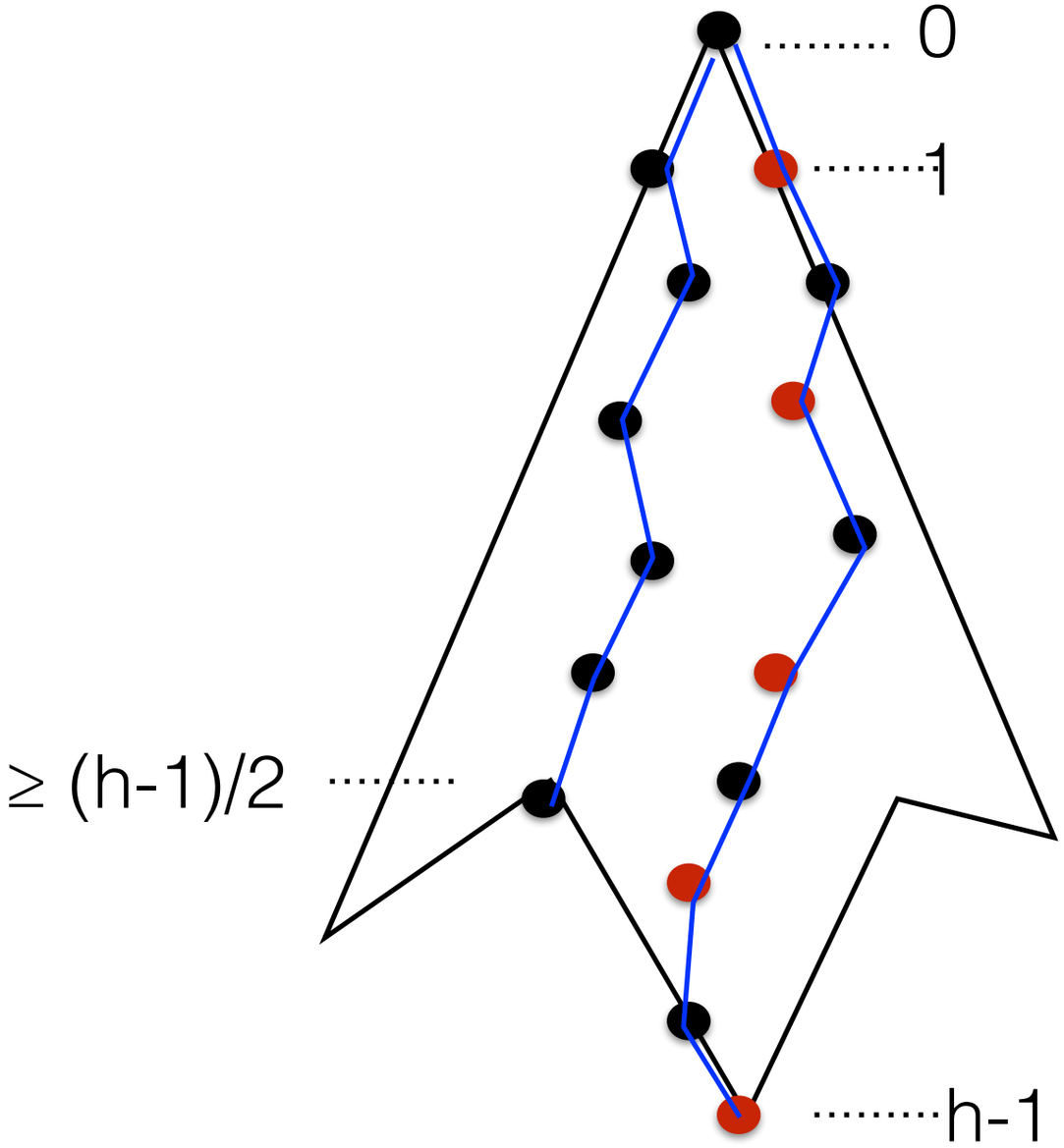
Rozszerzenia drzewa czerwono-czarnych

Definicja *Drzewo czerwono-czarne* to drzewo poszukiwań binarnych, które spełnia następujące warunki

1. każdy węzeł mam przypisany kolor: czerwony lub czarny
2. korzeń jest czarny
3. liście są czarne; przyjmujemy wariant drzewa z wartownikami - liśćmi są wartownicy
4. czerwony węzeł nie może mieć czerwonego syna
5. na każdej ścieżce od korzenia do liści jest tyle samo czarnych węzłów

Stwierdzenie Drzewo czerwono-czarne posiadające *n* węzłów wewnętrznych ma wysokość nie większą niż $2 \cdot lg(n+1)$



$$n \geq 2^{(h-1)/2+1} - 1 \geq 2^{h/2} - 1$$

$$lg(n+1) \geq h/2$$

$$2\, lg(n+1) \geq h$$

Rozszerzenie

Do każdego węzła **`x`** dodajemy nowe pole **`x.size`**

`x.size` = ilość węzłów (wewnętrznych czyli nie licząc wartowników) w poddrzewie o korzeniu `x` (włączając x)

Zauważmy:

```
x.size = x.left.size + x.right.size +1
```

Co nam daje takie rozszerzenie, czyli jakie nowe algorytmy możemy podać wykorzystujące pole `size`

Jak wprowadzić takie rozszerzenie do drzew czerwono-czarnych, czyli jak aktualizować pole `size` po każdej operacji wstawienia i usunięcia węzła

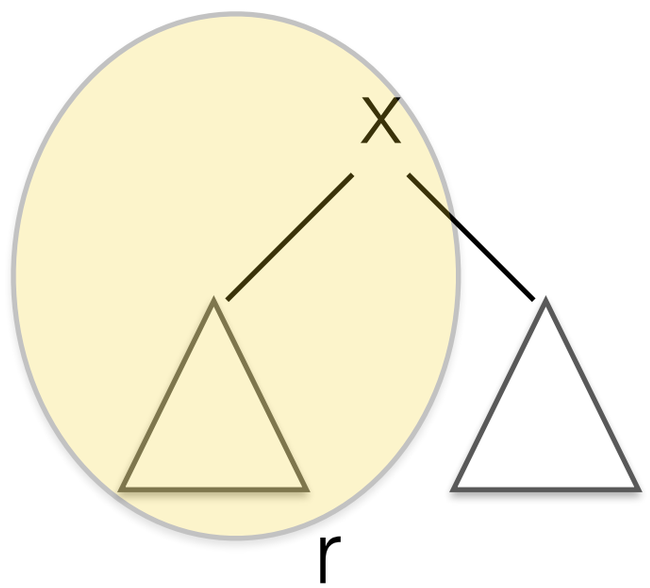
Wybór i-tego klucza (co do wartości)

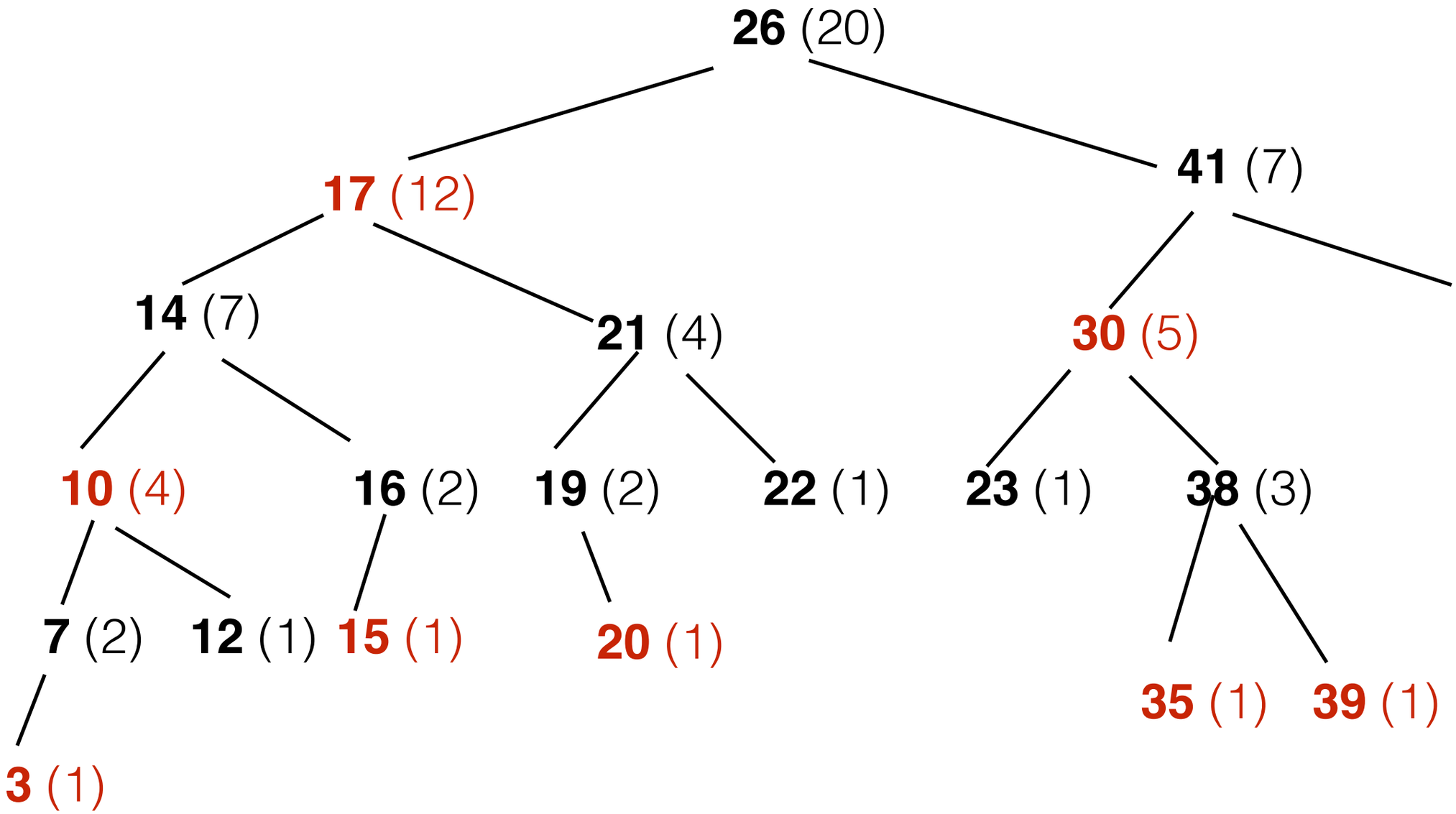
Dane: drzewo czerwono czarne rozszerzone o pole `size`, w którym wszystkie klucze są różne, oraz wartość `i,` taka że `1 ≤ i ≤ n,` gdzie `n` to ilość węzłów w drzewie

Wynik: węzeł zawierający i-ty co do wartości klucz, licząc od najmniejszych

```
Select(x, i)
r = x.left.size +1   // x jest r-ty so do wielkości
                     // w poddrzewie o korzeniu x

if i == r
    return x
else if i<r
        return Select(x.left, i)
      else
        return Select(x.right, i-r)
```





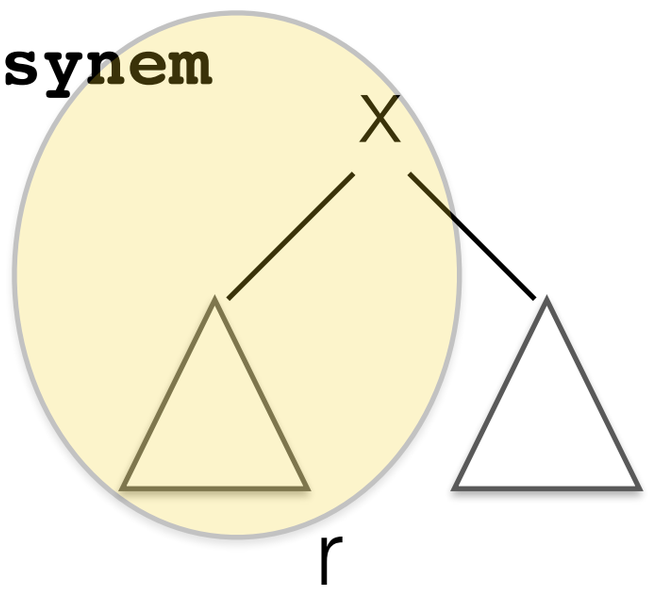

```
Select(26,17)    r = 13   < 17
Select(41,4)    r = 6   > 4
Select(30,4)    r = 2   < 4
Select(38,2)    r = 2
return 38
```

Ustalenie, który co do wartości jest dany węzeł w drzewie

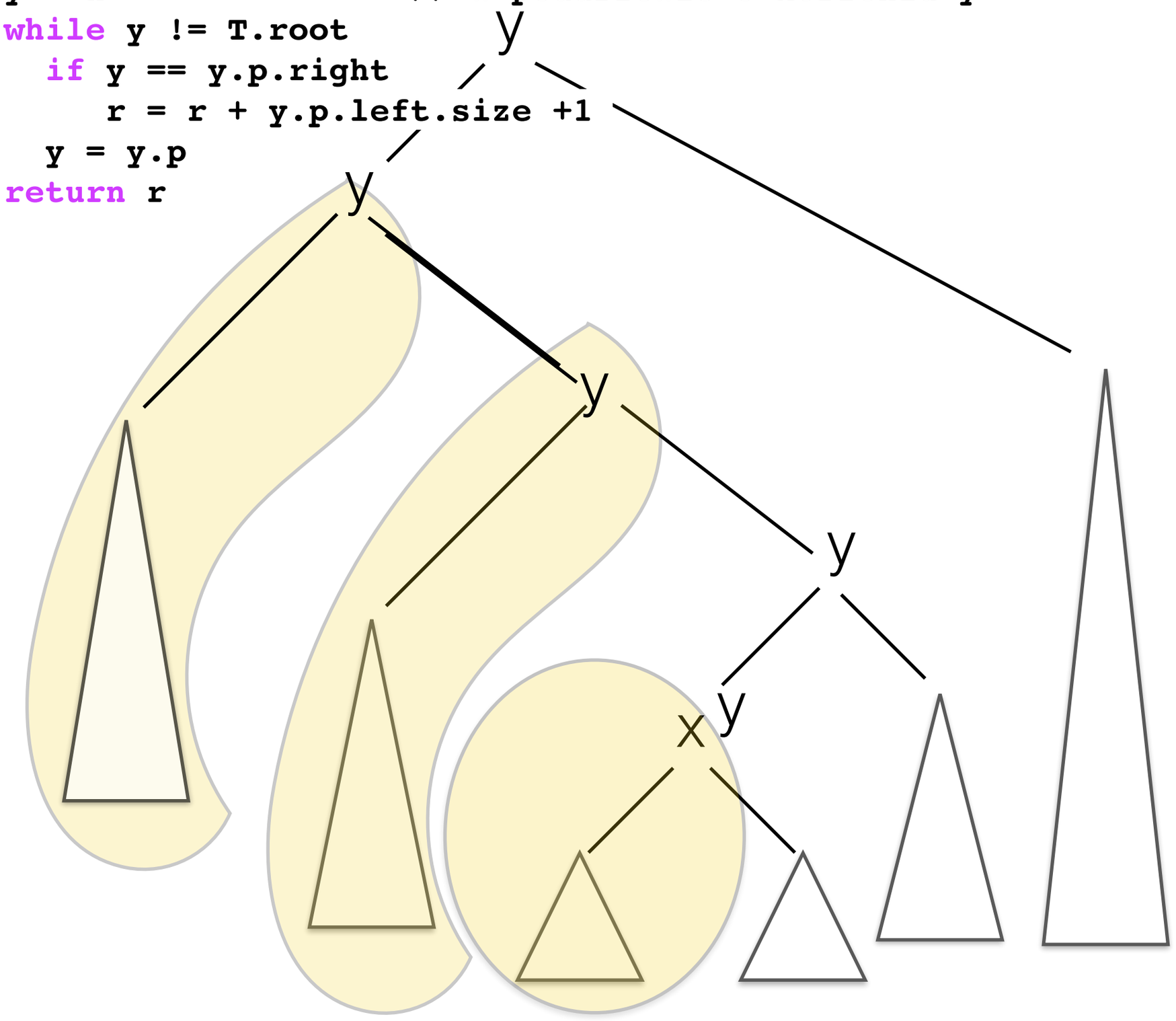
Dane: drzewo czerwono czarne T rozszerzone o pole `size`, w którym wszystkie klucze są różne, oraz węzeł x w tym drzewie

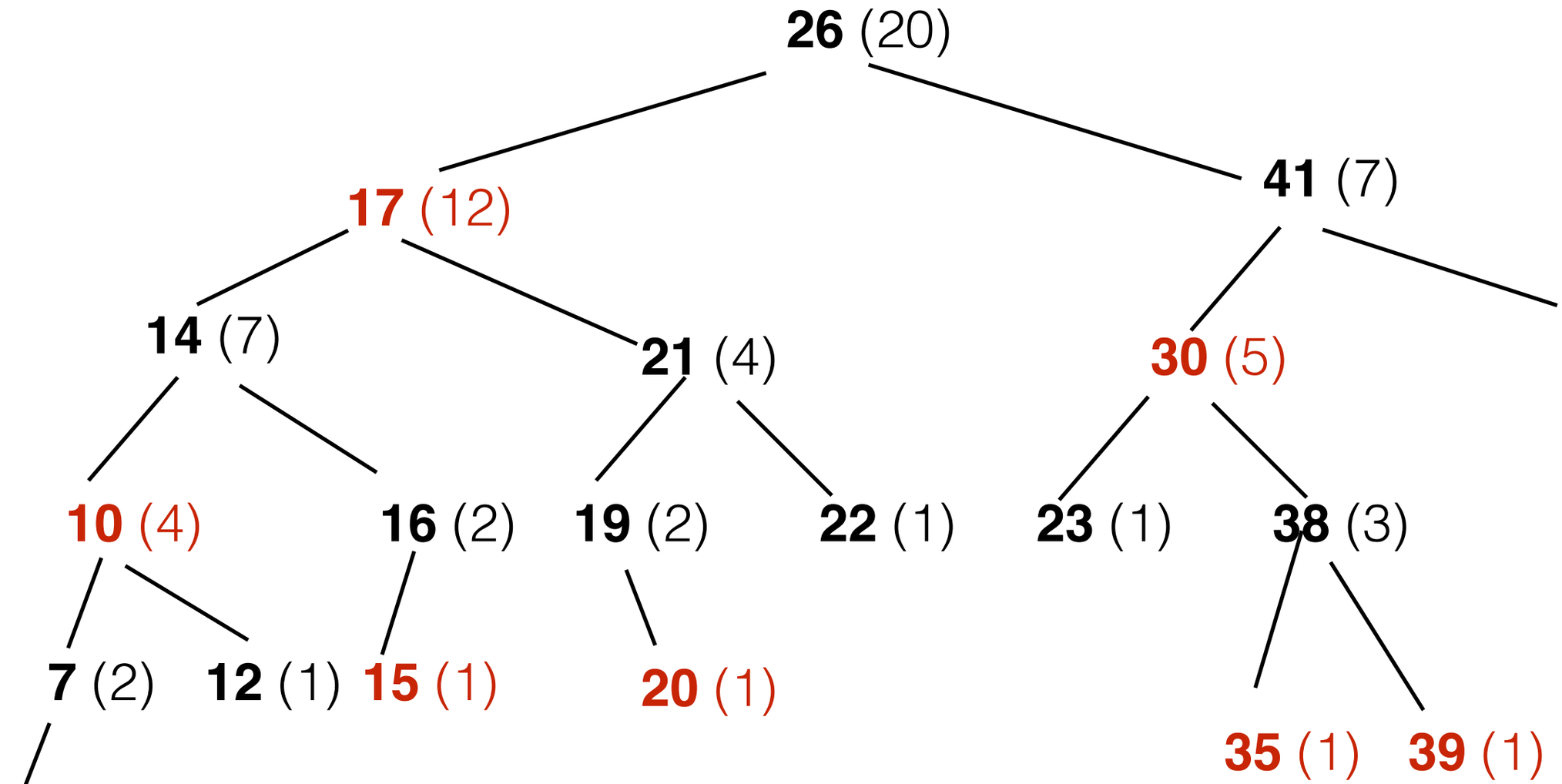
Wynik: który co do wartości, licząc od najmniejszych, jest w tym drzewie klucz zawarty w węźle x

```
Rank(T,x)
r = x.left.size + 1 // r == ilość kluczy <= x
y = x               // w poddrzewie o korzeniu y
while y != T.root
   if y == y.p.right  // y jest prawym synem
      r = r + y.p.left.size +1
   y = y.p
return r
```



```
Rank(T,x)
   r = x.left.size + 1 // r == ilość kluczy <= x
   y = x               // w poddrzewie o korzeniu y
   while y != T.root
     if y == y.p.right
        r = r + y.p.left.size +1
     y = y.p
   return r
```



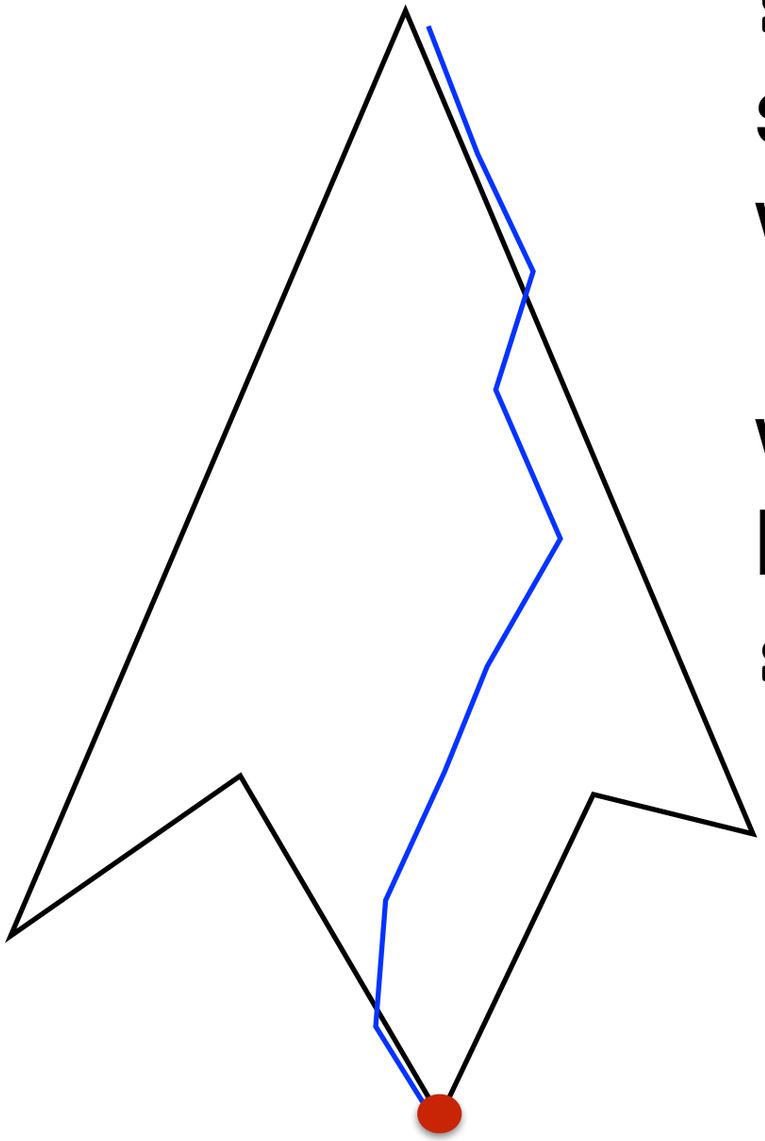



```
Rank(T,y)
x = y = 38    r = 2          y prawy
y = 30    r = r+2 =4         y lewy
y = 41    r = 4              y prawy
y = 26    r =  r + 12 + 1 = 17 root
return 17
```

Stwierdzenie Algorytmy `Select` i `Rank` mają czas pesymistyczny $\Theta(lg\ n)$.

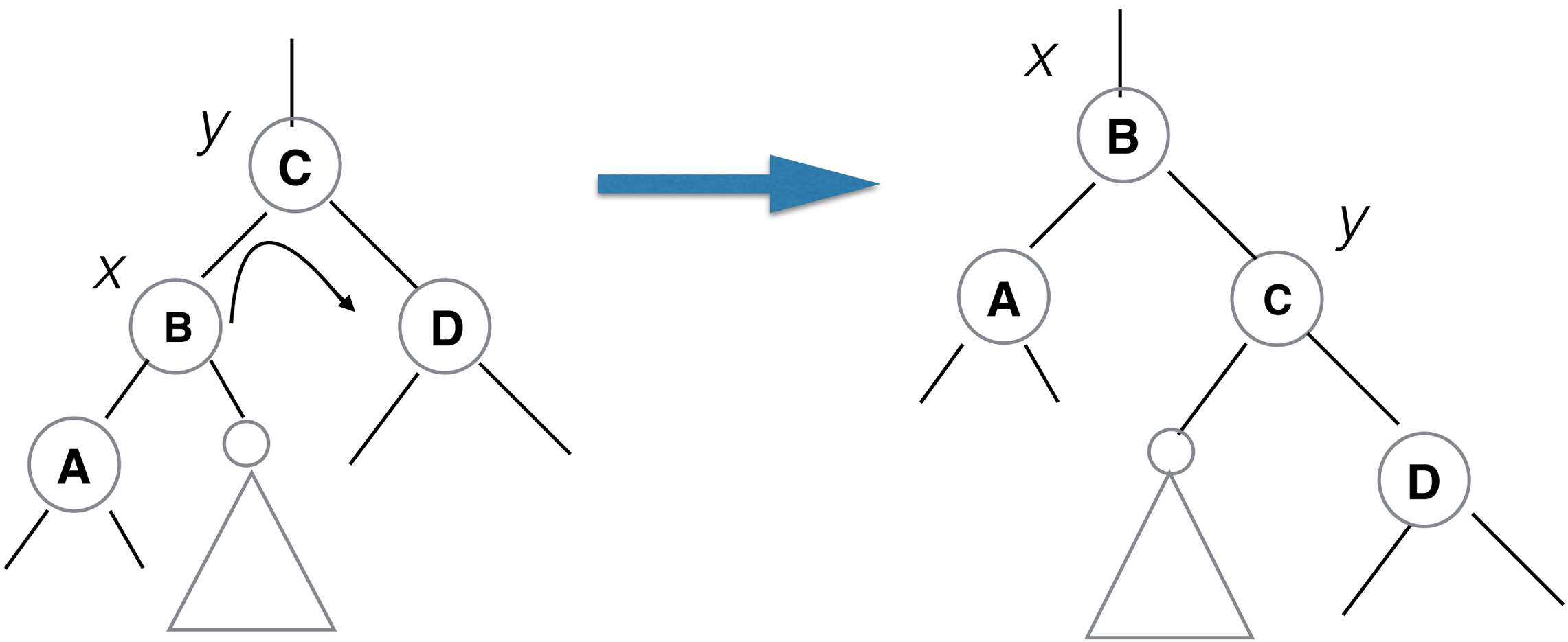
Uzasadnienie: drzewo cz-cz ma wysokość $\Theta(lg\ n)$

Aktualizacja pole `size` po każdej operacji wstawienia.



wstawiając węzeł zwiększamy pole `size` o 1 każdemu węzłowi na ścieżce od korzenia do wstawionego węzła

w czasie naprawiania węzła po każdej rotacji aktualizujemy pole `size`

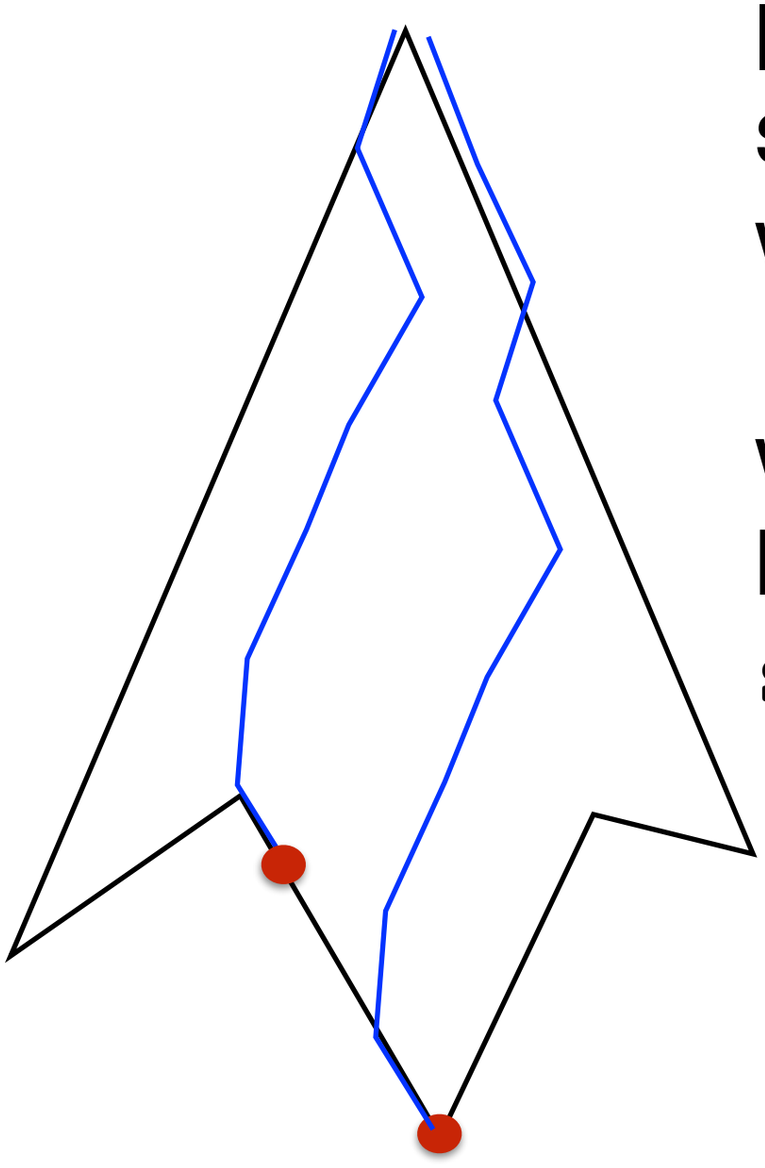


Po rotacji należy poprawić wartości **`size`**:

```
y.size = y.left.size + y.right.size +1
x.size = x.left.size + x.right.size +1
```

w pozostałych węzłach **`size`** się nie zmieniło

Aktualizacja pole `size` po każdej operacji usunięcia.



po usunięciu węzeła zmniejszamy pole `size` o 1 każdemu węzłowi na ścieżce od korzenia do usuniętego węzła

w czasie naprawiania węzła po każdej rotacji aktualizujemy pole `size`