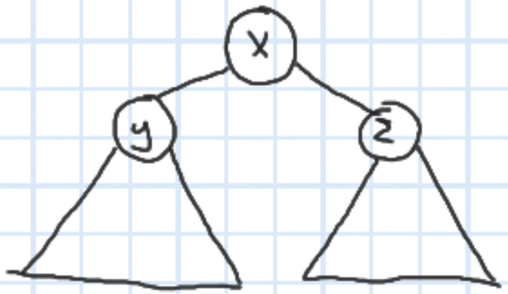
# Algorytmy i Struktury Danych
## Wykład 14

Tablice asocjacyjne – "coś co pozwala na indeksowanie dowolnym typem danych"

Drzewo BST (ang. binary-search tree)



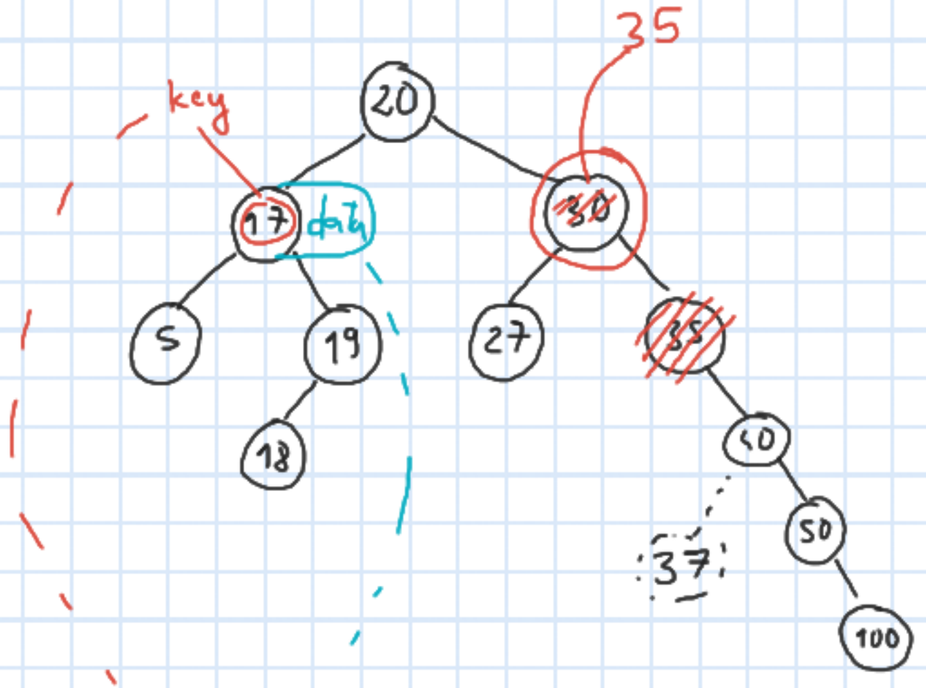

podstawowa zasada:

$y \leq x \leq z$

### Operacje

- search
- insert
- remove
- min / max
- pred / succ

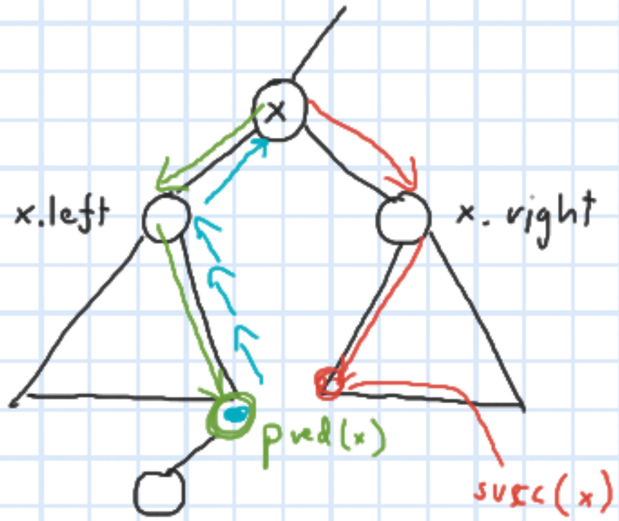
### Przykład




```
class BSTNode:
    def __init__(self):
        self.left = self.right = None
        self.parent = None
        self.key = None
        self.data = None
```

```
def search(root, key):
    while root ≠ None:
        if root.key == key: return root
        elif key < root.key: root = root.left
        else: root = root.right
    return None
```

# Obliczanie następnika i poprzednika w drzewie BST

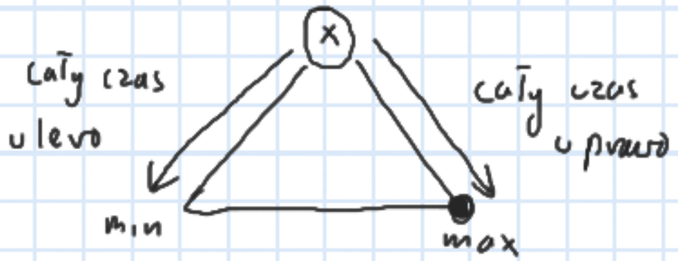



succ(x) – jeśli x ma prawe dziecko, to znajdź minimum w prawym poddrzewie x

- jeśli x nie ma prawego dziecka, to wędruj w górę drzewa, póki jesteś prawym synem

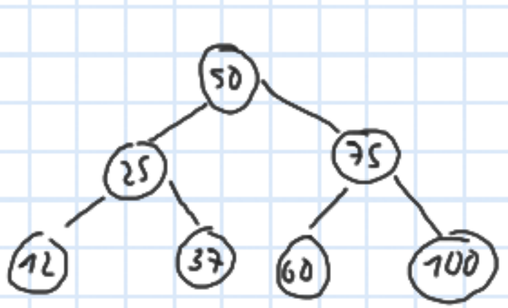
pred(x) – analogicznie

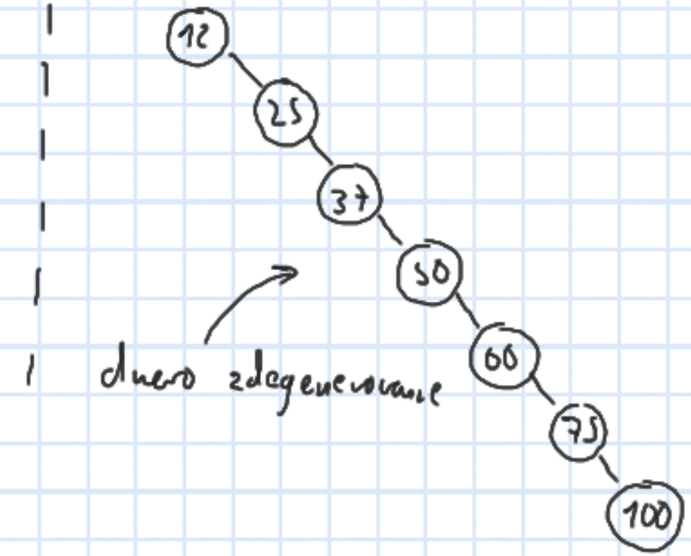
# Wyszukiwanie min/max w drzewie BST




## Złożoność

O(h) – gdzie h to wysokość drzewa

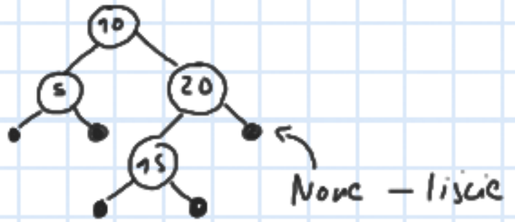


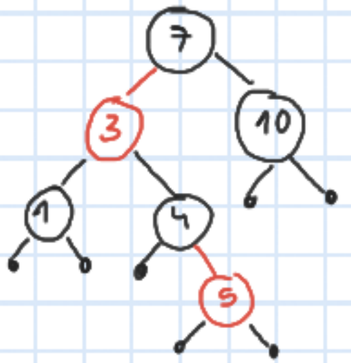
# Drzewa czerwono-czarne

Są to drzewa BST, w których dodatkowo obowiązują następujące zasady:

1. każdy węzeł jest albo czerwony, albo czarny
2. korzeń jest czarny
3. każdy liść jest czarny
4. jeśli węzeł jest czerwony, to obaj jego synowie są czarni
5. każda prosta ścieżka z ustalonego węzła do liścia zawiera tyle samo czarnych węzłów




## Przykład




$H(x)$ – wysokość drzewa zakorzenionego w $x$

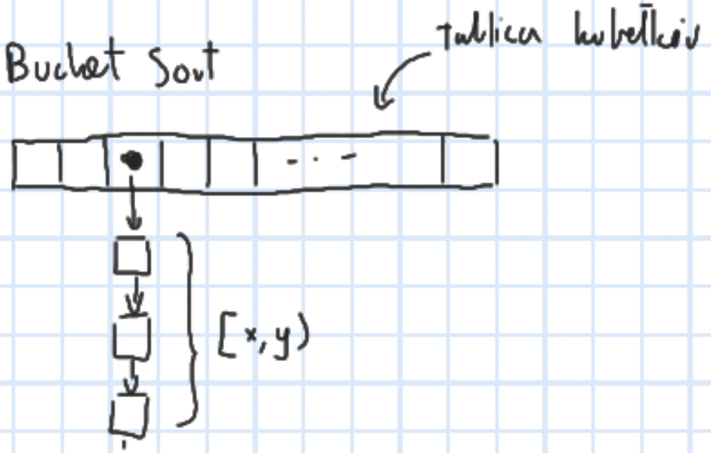
$BH(x)$ – j.w., ale liczy tylko czarne węzły (czarna wysokość)

$H(x) \leq 2BH(x)$

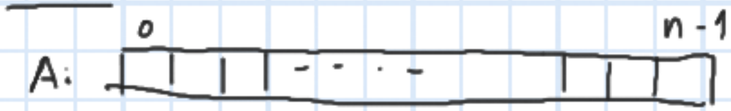
Drzewo zakorzenione w $x$ ma co najmniej $2^{BH(x)} - 1$ węzłów

# Tablice haszujące

Bucket Sort




Idea




Tworzymy funkcję haszującą, wyznaczającą indeks danego elementu

$h : \text{Dane} \longrightarrow \mathbb{N}$ (Dane: klucze)

Element o kluczu $d$ umieszczamy pod indeksem

$$h(d) \bmod n$$

Założenie: podobne klucze mają bardzo różne wartości funkcji haszującej

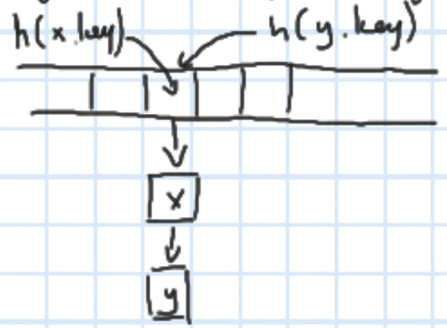
Co jeśli dla pewnych $x$ i $y$ zachodzi:

$$h(x.key) \equiv h(y.key) \pmod n$$

## Rozwiązywanie konfliktów

↳ metoda listowa

tablica haszująca jest tablicą list; dwa elementy o tym samym haszu przechowywane są na jednej liście

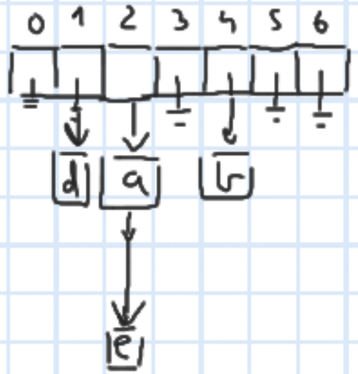



Bardziej skomplikowane usuwanie elementów z tablicy

↳ adresowanie otwarte

- wszystkie klucze bezpośrednio w tablicy
- jeśli dane pole jest zajęte, to dla nowego klucza znajdujemy inne
  - np. następne $h(x.key) + at \pmod n$ ← liczba prób
  - $h(x.key) + t\,h_2(x.key)$

# Przykład rozwiązywania konfliktów




a, b, c, d, e

$h(a) \bmod 7 = 2$

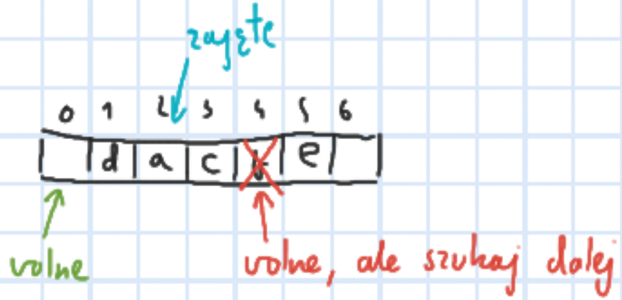
$h(b) \bmod 7 = 4$

$h(c) \bmod 7 = 2$

$h(d) \bmod 7 = 1$

$h(e) \bmod 7 = 2$

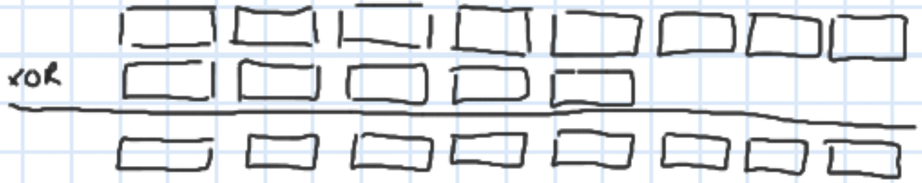
} listowe




} otwarte, liniowe

# Jak stworzyć funkcję haszującą?

- zamiana danych na liczę




- haszowanie uniwersalne

p - liczba pierwsza

$k \in \{0, \dots, p-1\}$

$$h(k) = \left[ (ak + b) \bmod p \right] \bmod n$$

wylosowany losowo