Algorytmy i SD

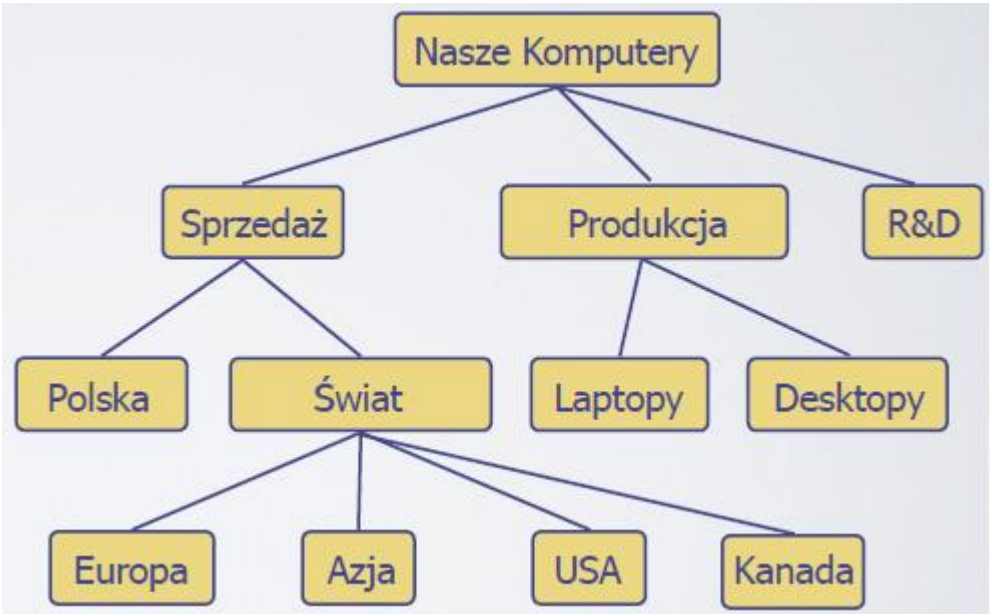
Struktury danych - Drzewa

Piotr Ciskowski, Łukasz Jeleń

Wrocław, 2023

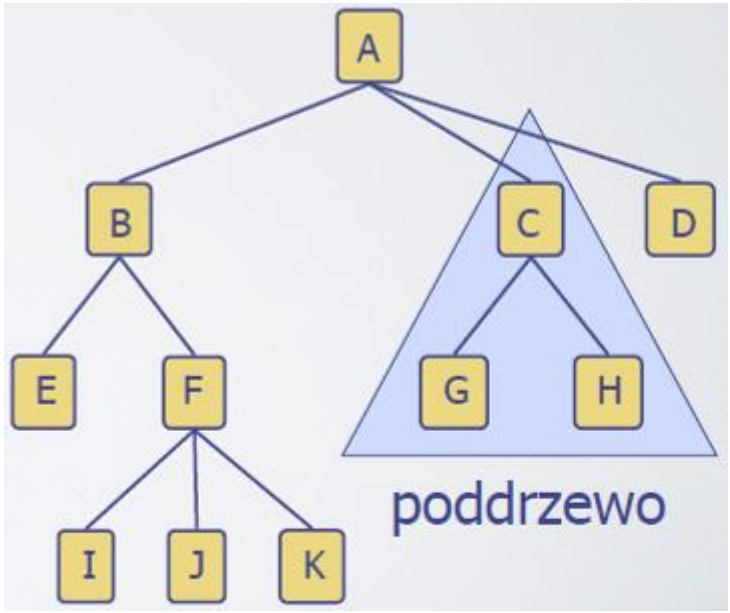
ADT **drzewo:**

- abstrakcyjny model struktury hierarchicznej
- składa się z węzłów ociec – syn
- zastosowania:
  - schematy organizacyjne
  - systemy plików
  - środowiska programistyczne



rysunek: Łukasz Jeleń

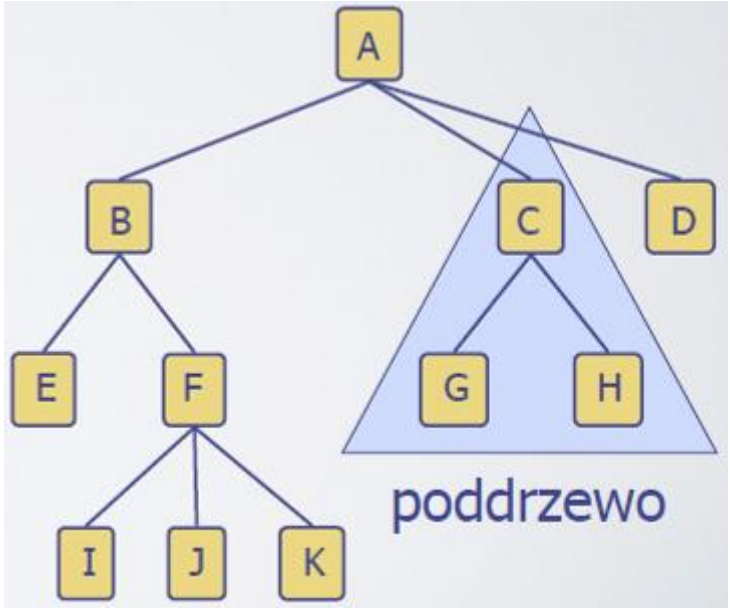
ADT **drzewo:**



- korzeń – węzeł nieposiadający rodzica (A)
- węzeł wewnętrzny – węzeł z przynajmniej jednym synem (A,B,C,F)
- węzeł zewnętrzny (a.k.a. liść) – węzeł bez dzieci (E,I,J,K,G,H,D)
- przodek węzła – ojciec, dziadek, pradziadek, itd.
- potomek węzła – syn, wnuk, prawnuk, itd.

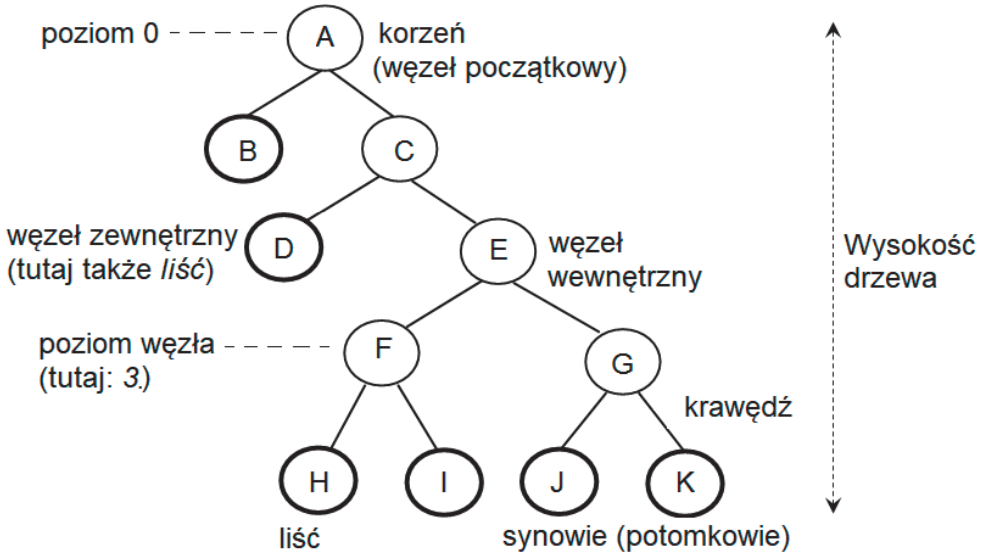
ADT **drzewo:**

- poziom węzła – liczba przodków
  liczba krawędzi
- wysokość drzewa – największy poziom występujący w drzewie (tu: 3)
- poddrzewo – węzeł i jego potomkowie



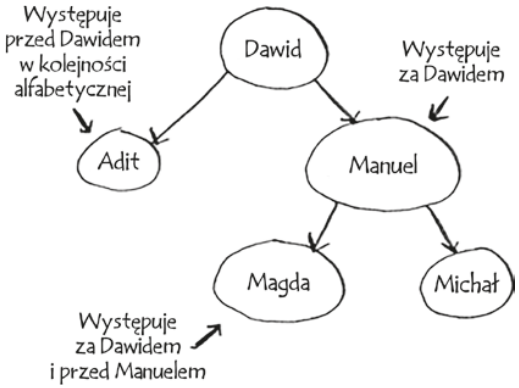
rysunek: Łukasz Jeleń

struktury danych – drzewo



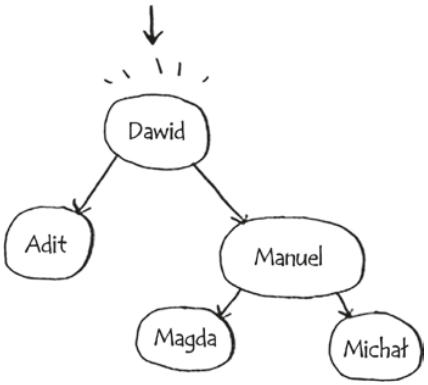
rysunek: Piotr Wróblewski. Algorytmy w Pythonie

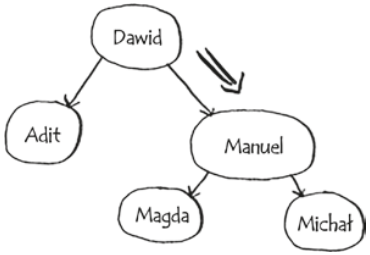
struktury danych – drzewo BST

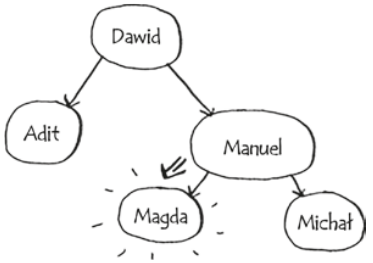


rysunek: Piotr Wróblewski. Algorytmy w Pythonie

struktury danych – drzewo BST







	Tablica	Binarne drzewo poszukiwań
Szukanie	O(log n)	O(log n)
Wstawianie	O(n)	O(log n)
Usuwanie	O(n)	O(log n)

struktury danych – drzewo

ELEMENT LISTY DWUKIERUNKOWEJ DANYCH	ELEMENT DRZEWA BINARNEGO (WĘZEŁ)

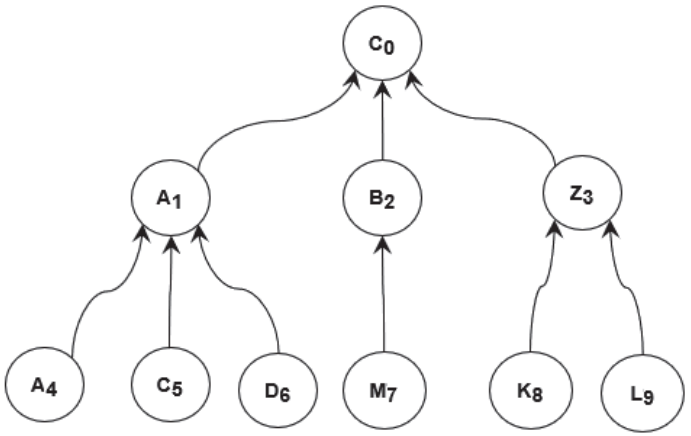
`class Element:`
`  def __init__(self, pInfo, ...):`
`  self.info = pInfo # Jakieś dane`
`  ...`
`  self.nastepny = None`
`  self.poprzedni = None`
	`class Wezel:`
	`  def __init__(self, pInfo, ...):`
	`  self.info = pInfo # Jakieś dane`
	`  ...`
	`  self.lewy = None`
	`  self.prawy = None`

```
class Wezel:
    def __init__(self, pInfo, ...):
    self.info = pInfo

    self.lewy = None      # Węzły potomne
    self.lewy = None      # Węzły potomne
    self.przodek = None # „Ojciec"
```

struktury danych – drzewo

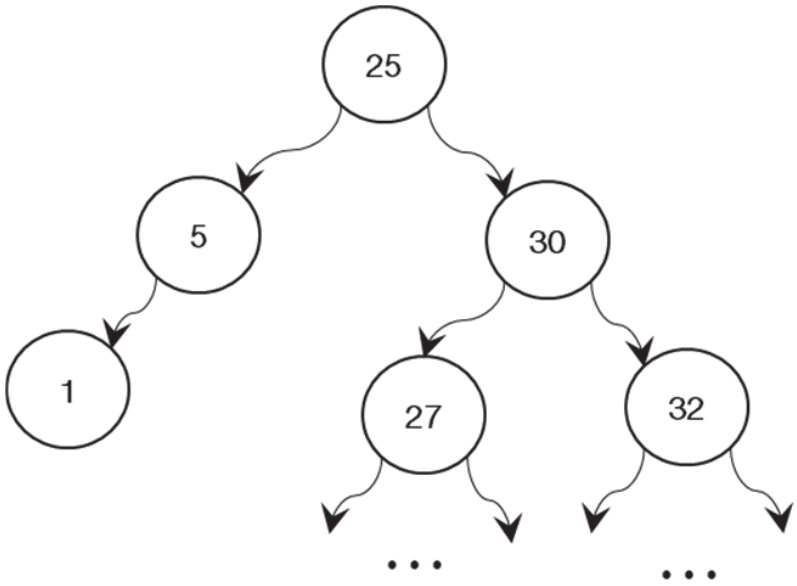
ELEMENT DRZEWA BINARNEGO (WĘZEŁ)

```
class Wezel:
    def __init__(self, pInfo, ...):
    self.info = pInfo # Jakieś dane
    ...
    self.lewy = None
    self.prawy = None
```



	syn	ojciec
0	C	0
1	A	0
2	B	0
3	Z	0
4	A	1
5	C	1
6	D	1
7	M	2
8	K	3
9	L	3

struktury danych – drzewo BST




```
class Wezel:
    def __init__(self, pKlucz=None):
        self.klucz = pKlucz    # Tutaj zapamiętamy klucz
        # Tutaj możemy dołożyć inne atrybuty (dane biznesowe)
        self.lewy = None     # Lewy potomek
        self.prawy = None    # Prawy potomek
```

struktury danych – drzewo BST

```
class BST:                      # Binarne drzewo poszukiwań – klasa może być dalej rozwijana
    def __init__(self):         # przy ukryciu detali realizacji klasy i służąc jako interfejs
                                # dostępowy do węzłów danych < tu ew. inne metaatrybuty >
        self.korzen = None      # Korzeń główny drzewa BST

    def szukaj(self, x):        # Zwraca węzeł o kluczu 'x' lub None
        if self.korzen == None:
            return None
        tmp = self.korzen
        while tmp.klucz != x:
            if x < tmp.klucz: # Kieruj się na lewo
                tmp = tmp.lewy
            else:             # Kieruj się na prawo
                tmp = tmp.prawy
            if tmp == None:   # Brak potomka
                return None
        return tmp            # Znaleziono
```

```
def Min(self):                  # Odszukaj i zwróć węzeł o najmniejszej wartości klucza
    tmp = self.korzen
    while tmp.lewy != None:     # Idź w lewo, aż do końca
        tmp = tmp.lewy
    return tmp
```

```
def Max(self):                  # Odszukaj i zwróć węzeł o największej wartości klucza
    tmp = self.korzen
    while tmp.prawy != None:    # Idź w prawo, aż do końca
        tmp = tmp.prawy
    return tmp
```

```
def wstaw(self, k):                 # Wersja iteracyjna wstawiania węzła do drzewa BST
    w = Wezel(k)                    # Tworzymy nowy węzeł i szukamy miejsca ma wstawienie
    if self.korzen == None:         # Jakoś pusto tutaj (na razie!)
        self.korzen = w
    else:                   # Coś tam jest, zatem szukamy aż do końca świata i jeden dzień dłużej!
        tmp = self.korzen
        while True:                 # Z pętli wyjdziemy po wstawieniu elementu (*)
            rodzic = tmp
            if k < tmp.klucz:       # Na lewo
                tmp = tmp.lewy
                if tmp == None:     # Jeśli koniec ścieżki, to wstaw na lewo
                    rodzic.lewy = w
                    break
            else:                   # Kierujemy się na prawo
                tmp = tmp.prawy
                if tmp == None:     # Jeśli koniec ścieżki, to wstaw na prawo
                    rodzic.prawy = w
                    break
        # Koniec pętli oznaczonej (*)
```

Skasowanie węzła wymaga przejścia przez przypadki:

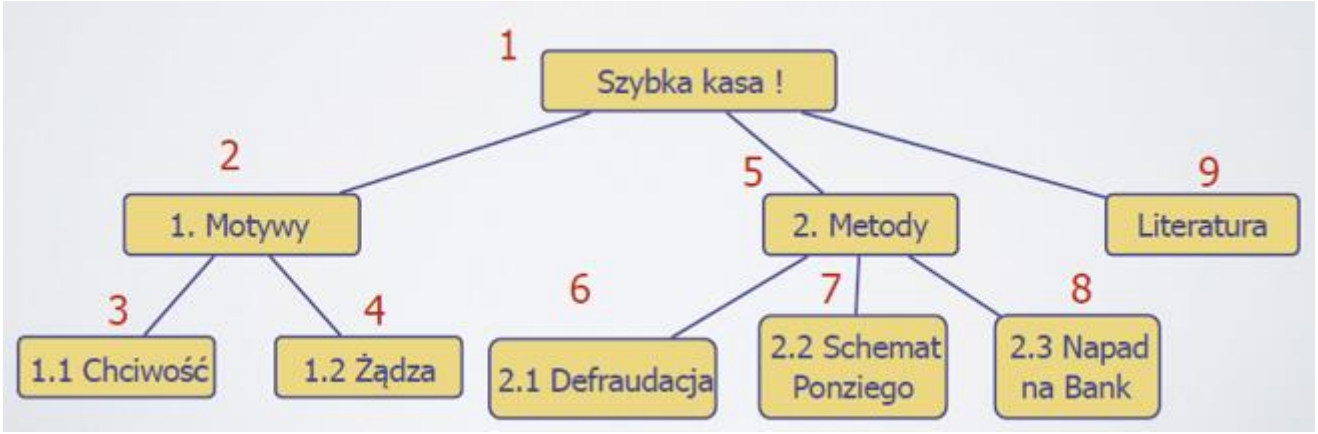
- Węzeł pusty (przypadek trywialny).
- Usuwanie liści drzewa (przypadek trywialny — po prostu kasujemy węzeł, pod nim już nic nie ma).
- Usuwanie węzła posiadającego jednego „potomka".
- Usuwanie węzła posiadającego dwóch „potomków".

```
def usunWezel(wierzcholek, klucz):              # Startujemy od węzła „wierzcholek"
    if wierzcholek == None:
        return wierzcholek
    if klucz < wierzcholek.klucz:               # Idziemy na lewo
        wierzcholek.lewy = usunWezel(wierzcholek.lewy, klucz)
    elif (klucz > wierzcholek.klucz):           # Idziemy na prawo
        wierzcholek.prawy = usunWezel(wierzcholek.prawy, klucz)
    else:  # Usuwamy znaleziony wierzchołek
        if wierzcholek.lewy==None:              # Wierzchołek z tylko jednym potomkiem
            temp = wierzcholek.prawy
            return temp
        elif wierzcholek.prawy==None:
            temp = wierzcholek.lewy
            return temp
        # Wierzchołek z dwoma potomkami => szukamy następcy kasowanego węzła,
        # który znajduje się w prawej gałęzi:
        temp = MinWezel(wierzcholek.prawy)
        # Kopiujemy zawartość następnika w miejsce usuwanego węzła (klucz i ew. inne atrybuty):
        wierzcholek.klucz = temp.klucz
        wierzcholek.prawy=usunWezel(wierzcholek.prawy,temp.klucz)# Usuwamy następnik
                                                                 # z podgałęzi
    return wierzcholek
```

```
def MinWezel(start):
    tmp = start
    while (tmp.lewy != None): # Idziemy skrajnie na lewo!
        tmp = tmp.lewy
    return tmp
```

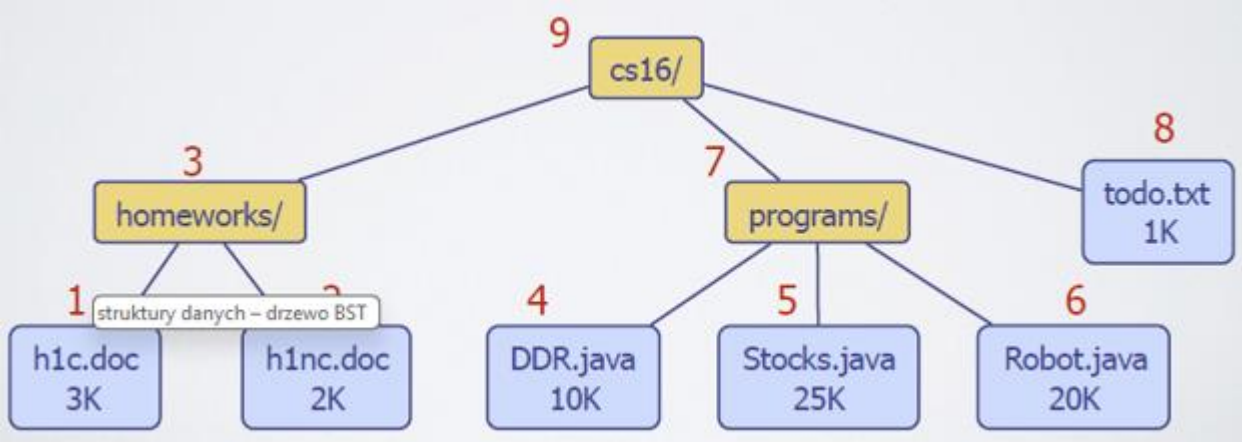
- przejścia – odwiedzają węzły w usystematyzowany sposób
- przejścia drzewa – **wzdłużne** – *pre-order*
  - węzeł jest odwiedzany przed jego potomkami
  - zastosowanie: wydruk hierarchicznej struktury czegoś

Algorytm *preOrder*(*v*)
  visit(*v*)
  for każdy syn *w* węzła *v*
    preOrder(*w*)



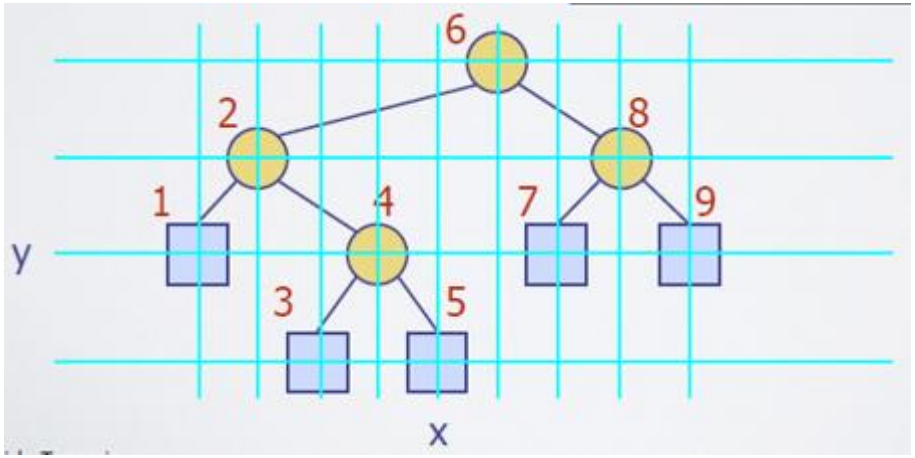
- przejścia – odwiedzają węzły w usystematyzowany sposób
- przejścia drzewa – **wsteczne** – *post-order*
  - węzeł jest odwiedzany po swoich potomkach
  - zastosowanie: obliczenie miejsca użytego przez pliki w katalogu wraz z jego podkatalogami

Algorytm *postOrder*(*v*)
  for każdy syn *w* węzła *v*
    postOrder(*w*)
  visit(*v*)



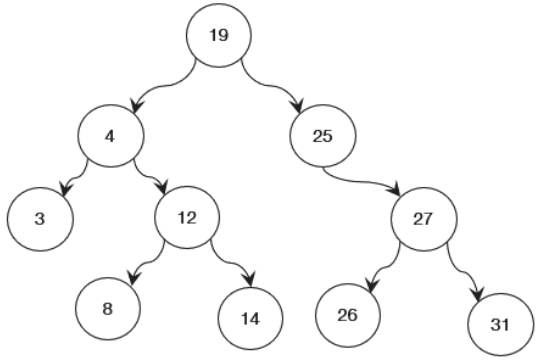
- przejścia – odwiedzają węzły w usystematyzowany sposób
- przejścia drzewa – **poprzeczne** – *in-order*
  - węzeł jest odwiedzany
    po Lewym, a przed Messim ;-)
  - zastosowanie: rysowanie drzewa binarnego

```
Algorytm inOrder(v)
  if hasLeft (v)
    inOrder (left (v))
  visit(v)
  if hasRight (v)
    inOrder (right (v))
```



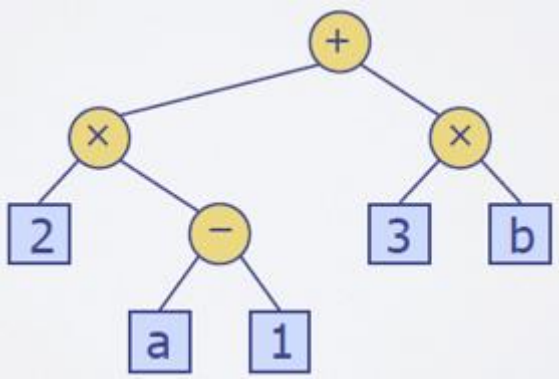
```
def preOrder(self, w):    # Przejście „wzdłużne"
    if w != None:
        print("[", w.klucz, "]", end=" ")
        self.preOrder(w.lewy)
        self.preOrder(w.prawy)

def inOrder(self, w):     # Przejście „poprzeczne"
    if w != None:
        self.inOrder(w.lewy)
        print("[", w.klucz, "]", end=" ")
        self.inOrder(w.prawy)

def postOrder(self, w):   # Przejście „wsteczne"
    if w != None:
        self.postOrder(w.lewy)
        self.postOrder(w.prawy)
        print("[", w.klucz, "]", end=" ")
```

struktury danych – drzewo BST



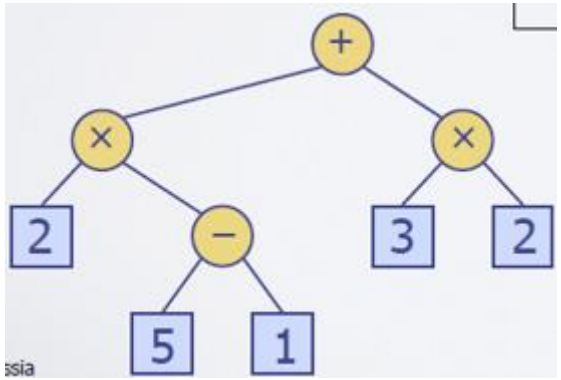
Pre-order:
19 →4 →3 →12 →8 →14 →25 →27 →26 →31
In-order:
3 →4 →8 →12 →14 →19 →25 →26 →27 →31
Post-order:
3 →8 →14 →12 →4 →26 →31 →27 →25 →19

- wyrażenia arytmetyczne
- drzewo binarne powiązane z operacją arytmetyczną
  - węzły wewnętrzne – operatory
  - węzły zewnętrzne – operandy
- przykład – drzewo operacji arytmetycznej
  dla wyrażenia: (2 × (*a* – 1) + (3 × *b*))



- wyznaczenie wartości – przejście wsteczne
  - metoda rekurencyjna zwracająca wartość poddrzewa
  - dla każdego węzła wewnętrznego łączymy wartości poddrzew

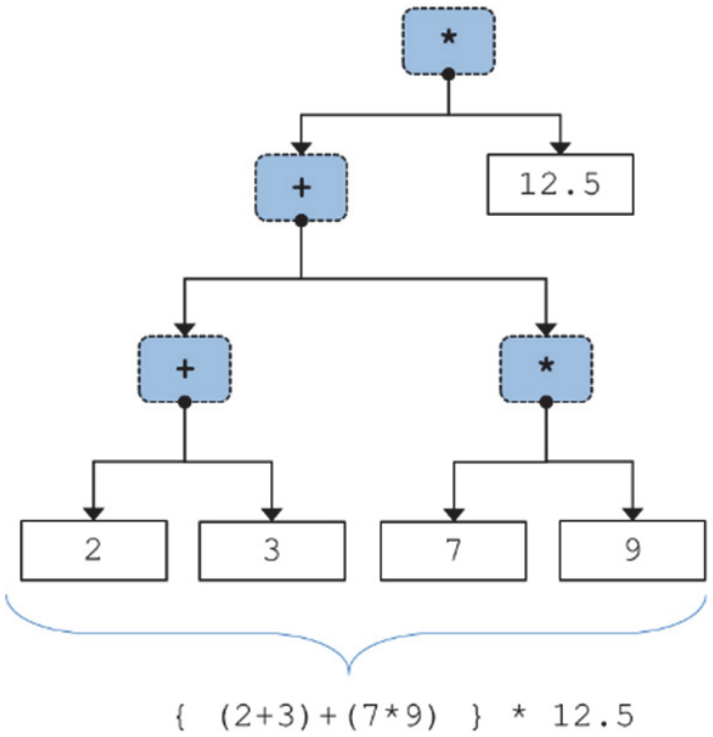
```
Algorytm evalExpr(T,v)
    if T.isExternal (T,v)
        return v.element ()
    else
        x ← evalExpr(T,T.left (v))
        y ← evalExpr(T,T.right (v))
        ◊ ← operator przechowywany w v
        return x ◊ y
```



```
def poprawne(self):   # Czy wyrażenie jest poprawne składniowo?
   if self.op=='0':
      return True     # Według naszej konwencji jest to liczba, więc akceptujemy
   if self.op in ['+', '-', '*', ':', '/']:    # Sprawdzimy teraz, czy jest nam znany
                                               # operator
         return (self.lewy).poprawne() and (self.prawy).poprawne()
   else:
      return False    # Błąd, nieznany operator lub inny błąd
```

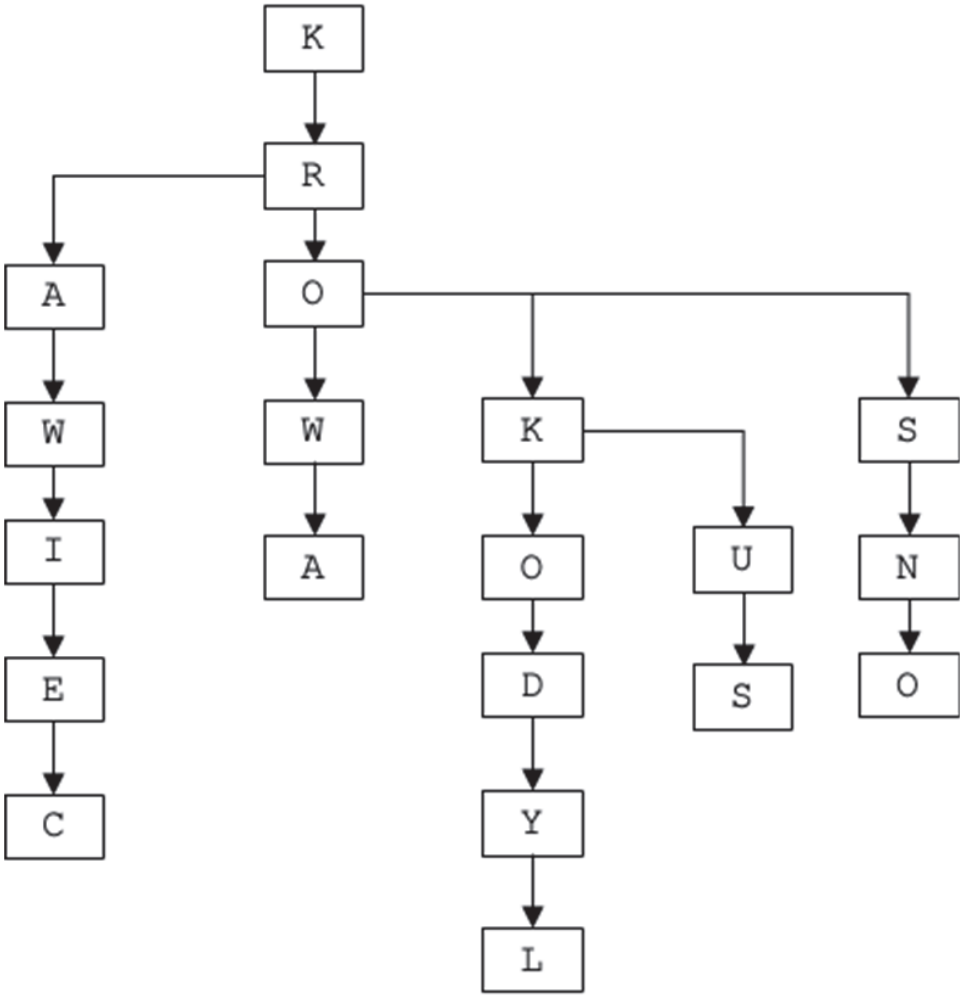
```
def oblicz(self):
   if self.poprawne():     # Wyrażenie poprawne?
      if (self.op=='0'):
         return self.val   # Pojedyncza wartość
      elif self.op=='+':
            return (self.lewy).oblicz()+(self.prawy).oblicz()
      elif self.op=='-':
         return (self.lewy).oblicz()-(self.prawy).oblicz()
      elif self.op=='*':
         return (self.lewy).oblicz()*(self.prawy).oblicz()
      elif (self.op==':' or self.op=='/'):
         if (self.prawy).oblicz()!= 0:
            return (self.lewy).oblicz() / (self.prawy).oblicz()
         else:
            print("\nDzielenie przez zero!")
            return -1      # Uproszczona sygnalizacja błędów
      else:
         print("Błąd składni")
         return -1            # Uproszczona sygnalizacja błędów
```

struktury danych – drzewo – wyrażenia arytmetyczne

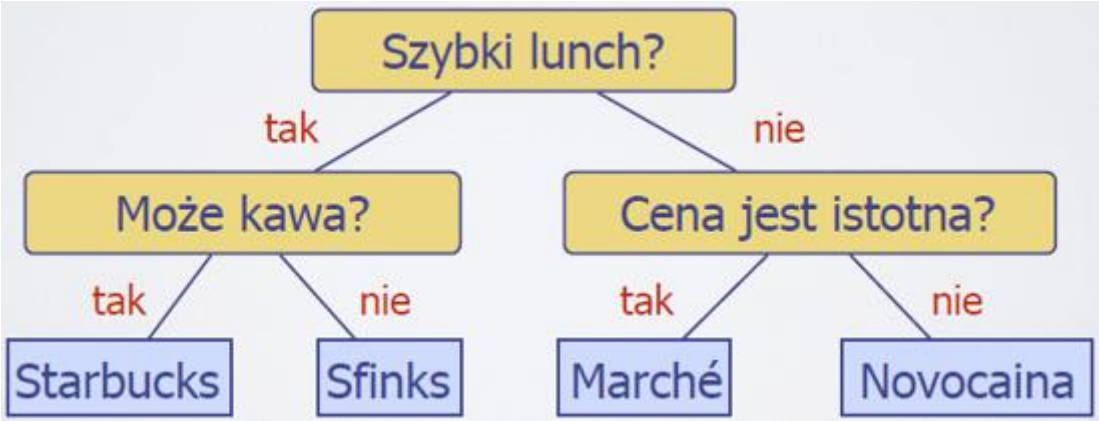



```
Prefiks:
*  12.5 +  *  9 7 +  3 2
Wynik: 850.0
Infiks:
(12.5 * ((9 * 7 )+ (3 + 2 )))
```

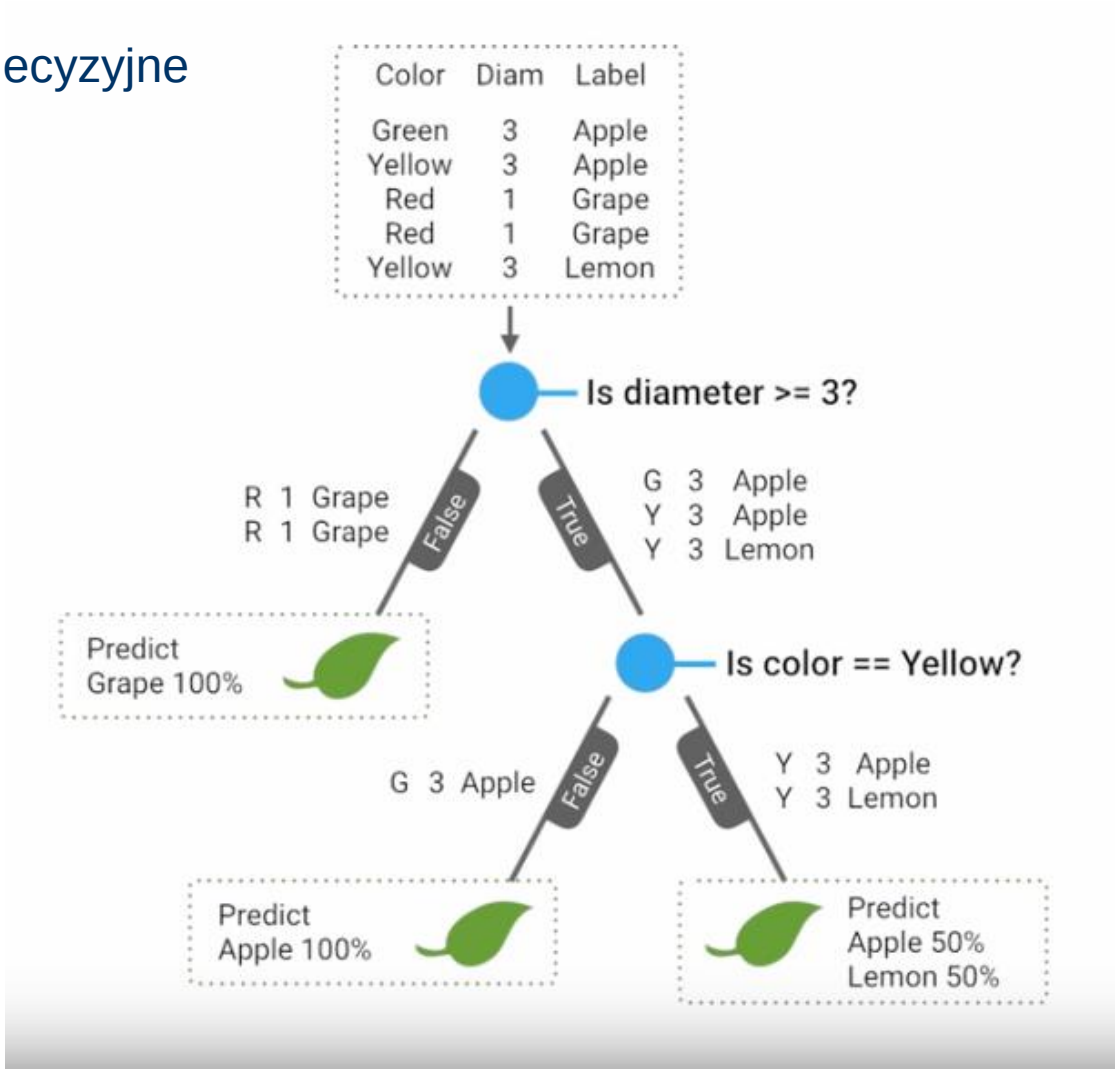
struktury danych – drzewo – USS



- drzewa decyzyjne
  - drzewo binarne związane z procesem decyzyjnym
  - węzły wewnętrzne – pytania o odpowiedziach tak/nie
  - węzły zewnętrzne – decyzje
  - przykład – decyzja dotycząca posiłku



- drzewa decyzyjne



rysunek: Josh Gordon

struktury danych – drzewo

- drzewa decyzyjne