https://github.com/KacperBudnik/AiSD

Zadanie 1

In [735…

```python
class TreeNode:
    def __init__(self,key,val,left=None,right=None, parent=None):
        self.key = key
        self.values=[] # do przechowywania dodatkowych wartości
        self.payload = val
        self.leftChild = left
        self.rightChild = right
        self.parent = parent
        self.size=1

    def hasLeftChild(self):
        return self.leftChild

    def hasRightChild(self):
        return self.rightChild

    def isLeftChild(self):
        return self.parent and self.parent.leftChild == self

    def isRightChild(self):
        return self.parent and self.parent.rightChild == self

    def isRoot(self):
        return not self.parent

    def isLeaf(self):
        return not (self.rightChild or self.leftChild)

    def hasAnyChildren(self):
        return self.rightChild or self.leftChild

    def hasBothChildren(self):
        return self.rightChild and self.leftChild

    def replaceNodeData(self,key,values_array,value,lc,rc):
        self.key = key
        self.payload = value
        self.values = values_array
        self.size=1+len(values_array)
        self.leftChild = lc
        self.rightChild = rc
        if self.hasLeftChild():
            self.leftChild.parent = self
        if self.hasRightChild():
            self.rightChild.parent = self

############# Dodane metody ###############

    def __len__(self):
        return self.size

    def add(self, val):
        self.values.append(val)
        self.size+=1

    def _pop(self):
        self.size-=1
        temp=self.payload
        self.payload=self.values.pop(0)
        return temp

    def val(self): # zwraca wszystkie wartości
        return [self.payload]+self.values

    def get_keys(self):
        temp=[self.key]
        if self.hasLeftChild():
            temp.extend(self.leftChild.get_keys())
        if self.hasRightChild():
            temp.extend(self.rightChild.get_keys())
        return temp
```

In [761…

```python
class BinarySearchTree:

    def __init__(self, *start_value):
        self.root = None
        self.size = 0
        val=start_value
        if len(val)==1:
            if type(val[0]) is dict:
                for i in val[0]:
                    self.put(i,val[0][i])
            elif type(val[0]) is BinarySearchTree:
                for i in val[0]:
                    self.put(*i)
        elif len(val)==2:
            self.add(*val)
        elif len(val)==0:
            pass
        else:
            raise ValueError("Podono za dużo elementów")


    def length(self):
        return self.size

    def __len__(self):
        return self.size

    def put(self,key,val):
        if not type(key) in (float,int):
            raise KeyError("Podano niewłaściwy klucz")
        if self.root:
            self._put(key,val,self.root) #_put is a helper function
        else:
            self.root = TreeNode(key,val)
        self.size = self.size + 1

    def _put(self,key,val,currentNode):
        if key < currentNode.key:
            if currentNode.hasLeftChild():
                self._put(key,val,currentNode.leftChild)
            else:
                currentNode.leftChild = TreeNode(key,val,parent=currentNode)
        elif key > currentNode.key:
            if currentNode.hasRightChild():
                self._put(key,val,currentNode.rightChild)
            else:
                currentNode.rightChild = TreeNode(key,val,parent=currentNode)
        else:
            currentNode.add(val)

    def __setitem__(self,k,v): #overloading of [] operator
        self.put(k,v)

    def get(self,key):
        if self.root:
            res = self._get(key,self.root)
            if res:
                return res.payload
            else:
                return None
        else:
            return None

    def _get(self,key,currentNode):
        if not currentNode:
            return None
        elif currentNode.key == key:
            return currentNode
        elif key < currentNode.key:
            return self._get(key,currentNode.leftChild)
        else:
            return self._get(key,currentNode.rightChild)

    def __getitem__(self,key): #overloading of [] operator
        return self.get(key)

    def __contains__(self,key):  # overloading of in operator
        if self._get(key,self.root):
            return True
        else:
            return False




    def delete(self,key):
        if self.size > 1:
            nodeToRemove = self._get(key,self.root)
            if nodeToRemove:
                self.remove(nodeToRemove)
                self.size = self.size-1
            else:
                raise KeyError('Error, key not in tree')
        elif self.size == 1 and self.root.key == key:
            self.root = None
            self.size = self.size - 1
        else:
            raise KeyError('Error, key not in tree')

    def __delitem__(self,key): #overloading of del operator
        self.delete(key)

    def spliceOut(self):
        if self.isLeaf():
            if self.isLeftChild():
                self.parent.leftChild = None
            else:
                self.parent.rightChild = None
        elif self.hasAnyChildren():
            if self.hasLeftChild():
                if self.isLeftChild():
                    self.parent.leftChild = self.leftChild
                else:
                    self.parent.rightChild = self.leftChild
                    self.leftChild.parent = self.parent
            else:
                if self.isLeftChild():
                    self.parent.leftChild = self.rightChild
                else:
                    self.parent.rightChild = self.rightChild
                self.rightChild.parent = self.parent

    def findSuccessor(self):
        succ = None
        if self.hasRightChild():
            succ = self.rightChild.findMin()
        else:
            if self.parent:
                if self.isLeftChild():
                     succ = self.parent
                else:
                    self.parent.rightChild = None
                    succ = self.parent.findSuccessor()
                    self.parent.rightChild = self
        return succ

    def findMin(self):
        current = self
        while current.hasLeftChild():
            current = current.leftChild
        return current

    def remove(self,currentNode):
        if len(currentNode)>1: # Jeśli jest parę wartości usuń ostatnią
            currentNode._pop()
        elif currentNode.isLeaf(): #leaf
            if currentNode == currentNode.parent.leftChild:
                currentNode.parent.leftChild = None
            else:
                currentNode.parent.rightChild = None
        elif currentNode.hasBothChildren(): #interior
            succ = currentNode.findSuccessor()
            succ.spliceOut()
            currentNode.key = succ.key
            currentNode.payload = succ.payload

        else: # this node has one child
            if currentNode.hasLeftChild():
                if currentNode.isLeftChild():
                    currentNode.leftChild.parent = currentNode.parent
                    currentNode.parent.leftChild = currentNode.leftChild
                elif currentNode.isRightChild():
                    currentNode.leftChild.parent = currentNode.parent
                    currentNode.parent.rightChild = currentNode.leftChild
                else:
                    currentNode.replaceNodeData(currentNode.leftChild.key,
                                    currentNode.leftChild.values,
                                    currentNode.leftChild.payload,
                                    currentNode.leftChild.leftChild,
                                    currentNode.leftChild.rightChild)
            else:
                if currentNode.isLeftChild():
                    currentNode.rightChild.parent = currentNode.parent
                    currentNode.parent.leftChild = currentNode.rightChild
                elif currentNode.isRightChild():
                    currentNode.rightChild.parent = currentNode.parent
                    currentNode.parent.rightChild = currentNode.rightChild
                else:
                    currentNode.replaceNodeData(currentNode.rightChild.key,
                                    currentNode.rightChild.values,
                                    currentNode.rightChild.payload,
                                    currentNode.rightChild.leftChild,
                                    currentNode.rightChild.rightChild)


###########################################

    def add(self, key, val):
        self.put(key,val)

    def values(self,k): # Zwraca listę wszystkich wartości dla danej liczby (na wypadek jeśli było podanych wię
        temp=self._get(k,self.root)
        if temp:
            return temp.val()
        else:
            raise KeyError("Podany klucz nie istnieje")

    def get_keys(self): # zwraca wszystkie klucze
        if self.root:
            return self.root.get_keys()
            #key_list=[self.root.key]
        else:
            return []

    def __iter__(self):
        return BinarySearchTreeIterator(self)


class BinarySearchTreeIterator: # Do iterowania zadanego drzewa
    def __init__(self, tree:BinarySearchTree):
        self.keys=tree.get_keys()
        self.index=-1
        self.max=len(self.keys)
        self.tree=tree


    def __next__(self):
        self.index+=1
        if self.index<self.max:
            return (self.keys[self.index],self.tree[self.keys[self.index]]) # zwraca tylko pierwszy dodany ele
        raise StopIteration
```

Tworzymy drzewko, z zadanymi warotściami początkowymi.

In [762…

```python
a=BinarySearchTree(1,"Pierwszy element")
```

In [763…

```python
a[1]
```

Out[763…

```
'Pierwszy element'
```

Dodajemy elemnty

In [764…

```python
a.add(3,"Drugi dodany element")
```

In [765…

```python
a.add(1,"Drugi dodany element do wartości 1")
```

In [766…

```python
a.add(1,"A tu trzeci!")
```

Sprawdzamy wartość 1

In [768…

```python
a[1]
```

Out[768…

```
'Pierwszy element'
```

Pozostałe zostały zapisane w `TreeNode`, ale nie są wyświetlane. By je wyświetlić piszemy

In [769…

```python
a.values(1)
```

Out[769…

```
['Pierwszy element', 'Drugi dodany element do wartości 1', 'A tu trzeci!']
```

Możemy teraz usunąć element i zobaczyć zmianę ilości elementów

In [770…

```python
a.delete(1)
```

In [771…

```python
a[1]
```

Out[771…

```
'Drugi dodany element do wartości 1'
```

In [774…

```python
a.values(1)
```

Out[774…

```
['Drugi dodany element do wartości 1', 'A tu trzeci!']
```

I ponownie

In [775…

```python
a.delete(1)
```

In [776…

```python
a.delete(1)
```

In [777…

```python
a[1]
```

In [778…

```python
a.values(1)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-778-643ae2bde184> in <module>
----> 1 a.values(1)

<ipython-input-761-b3773ede42fd> in values(self, k)
    194             return temp.val()
    195         else:
--> 196             raise KeyError("Podany klucz nie istnieje")
    197 
    198     def get_keys(self): # zwraca wszystkie klucze

KeyError: 'Podany klucz nie istnieje'
```

Znikło :c
Ale ta powinno być

Możemy też sprawdzić, czy nie ma 1 w kluczach

In [779…

```python
a.get_keys()
```

Out[779…

```
[3]
```

No nie ma, stwórzmyu nowe drzewo, ale od słownik

In [780…

```python
a=BinarySearchTree({2:3,12:"cos", 3.2:"moze"})
```

In [781…

```python
a.get_keys()
```

Out[781…

```
[2, 12, 3.2]
```

Iterator zwraca parę wartości `(key, payload)`

In [783…

```python
for i in a:
    print(i)
```

```
(2, 3)
(12, 'cos')
(3.2, 'moze')
```

In [784…

```python
a.add(52,"23")
```

In [785…

```python
for i in a:
    print(i)
```

```
(2, 3)
(12, 'cos')
(3.2, 'moze')
(52, '23')
```

In [786…

```python
a.add("tak", 12)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-786-66783ae2cb89> in <module>
----> 1 a.add("tak", 12)

<ipython-input-761-b3773ede42fd> in add(self, key, val)
    187 
    188     def add(self, key, val):
--> 189         self.put(key,val)
    190 
    191     def values(self,k): # Zwraca listę wszystkich wartości dla danej liczby (na wypadek jeśli było poda
nych więcej niż 1)

<ipython-input-761-b3773ede42fd> in put(self, key, val)
     28     def put(self,key,val):
     29         if not type(key) in (float,int):
---> 30             raise KeyError("Podano niewłaściwy klucz")
     31         if self.root:
     32             self._put(key,val,self.root) #_put is a helper function

KeyError: 'Podano niewłaściwy klucz'
```

Ale możemy dodać tylko liczby

In [787…

```python
a.get_keys()
```

Out[787…

```
[2, 12, 3.2, 52]
```

Ale za to możemy też tworzyć drzewo od drzewa

In [788…

```python
b=BinarySearchTree(a)
```

In [792…

```python
for i in b:
    print(i)
```

```
(2, 3)
(12, 'cos')
(3.2, 'moze')
(52, '23')
```

In [793…

```python
b.delete(52)
```

In [794…

```python
b[52]
```

In [795…

```python
a[52]
```

Out[795…

```
'23'
```

No i tak stworzone drzewo nie ingeruje w oryginał

Zadanie 2

In [204…

```python
import types
import numpy as np

class BinHeap:
    def __init__(self,*items): # skrócenie nazw + możliwość podania elementów kopca przy dodawaniu
        self.list = [0]
        self.size = 0
        self.add(*items)

    def percUp(self,i):
        while i // 2 > 0:
            if self.list[i] < self.list[i // 2]:
                tmp = self.list[i // 2]
                self.list[i // 2] = self.list[i]
                self.list[i] = tmp
            i = i // 2

    def insert(self,k):
        self.list.append(k)
        self.size = self.size + 1
        self.percUp(self.size)

    def findMin(self):
        if self.size==0:
            raise IndexError("Kopiec jest pusty")
        return self.list[1]

    def percDown(self,i):
        while (i * 2) <= self.size:
            mc = self.minChild(i)
            if self.list[i] > self.list[mc]:
                tmp = self.list[i]
                self.list[i] = self.list[mc]
                self.list[mc] = tmp
            i = mc

    def minChild(self,i):
        if i * 2 + 1 > self.size:
            return i * 2
        else:
            if self.list[i*2] < self.list[i*2+1]:
                return i * 2
            else:
                return i * 2 + 1

    def delMin(self):
        if self.size==0:
            raise IndexError("Kopiec jest pusty")
        retval = self.list[1]
        self.list[1] = self.list[self.size]
        self.size = self.size - 1
        self.list.pop()
        self.percDown(1)
        return retval

    def buildHeap(self,alist): # już nie potrzebne (jest to _clear i dodanie po prostu listy)
        i = len(alist) // 2
        self.size = len(alist)
        self.list = [0] + alist[:]
        while (i > 0):
            self.percDown(i)
            i = i - 1

    def size(self):
        return self.size

    def isEmpty(self):
        return self.size == 0

    def __str__(self):
        txt = "{}".format(self.list[1:])
        return txt

#################################################################
#                              Własne                           #
#################################################################

    def __repr__(self):
        return str(self.list[1:])

    def __len__(self):
        return self.size

    def add(self,*items): # by było można podawać wiele argumentów, wraz z kontenerami
        for item in items:
            if type(item) in (tuple, list):
                self.add(*item)
            elif type(item) in (dict, set):
                self.add(*list(item))
            elif type(item) == types.GeneratorType:
                for i in item:
                    self.add(i)
            elif isinstance(item, np.ndarray):
                item=list(item)
                self.add(item)
            else:
                self.insert(item)

    def _clear(self): # czyszczenie kopca (prywatna, by nie użyć przez przypadek)
        self.list=[0]
        self.size=0

    def sort(self):
        sortedList=[]
        s=self.size
        while self.size>0:
            sortedList.append(self.delMin())
        self.list=[0]+sortedList
        self.size=s
```

In [205…

```python
from random import randint
a=BinHeap(randint(-50,50) for i in range(31))
```

In [206…

```python
a
```

Out[206…

```
[-46, -39, -41, -33, -32, -27, -34, -33, -8, -32, -31, 16, -16, -13, 19, 50, 23, 43, 7, 38, -3, 41, 27, 39, 21,
37, 11, 38, 14, 43, 26]
```

In [207…

```python
a.sort()
```

In [208…

```python
a
```

Out[208…

```
[-46, -41, -39, -34, -33, -33, -32, -32, -31, -27, -16, -13, -8, -3, 7, 11, 14, 16, 19, 21, 23, 26, 27, 37, 38,
38, 39, 41, 43, 43, 50]
```

Co tu tłumaczyć, sortowanie przez kopcowanie

Test

Poniżej pojawiają się powiadomienia o błędach. Są to przerwania wykonywania iteracji. Zrobiłem delikatny błąd rachunkowy i zamiast planowanego wykonania programu w ciągu 8 godzin, wykonywałby się ponad 24. Nie puszczałem symulacji jeszcze raz, tylko ograniczyłem liczbę wyników to tych które zdążyły się wykonać

In [46]:

```python
import time
import numpy as np
from random import randint

n=31
times=np.zeros(n+1)
numbers=np.random.randint(-2**n,2**n,2**n)

for i in range(n+1):
    a=BinHeap(numbers[:2**i])
    T=time.time()
    a.sort()
    times[i]=time.time()-T
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-46-0a11c17ee2f6> in <module>
     10     a=BinHeap(numbers[:2**i])
     11     T=time.time()
---> 12     a.sort()
     13     times[i]=time.time()-T

<ipython-input-1-1dd12c072ab6> in sort(self)
    117         s=self.size
    118         while self.size>0:
--> 119             sortedList.append(self.delMin())
    120         self.list=[0]+sortedList
    121         self.size=s

<ipython-input-1-1dd12c072ab6> in delMin(self)
     52         self.size = self.size - 1
     53         self.list.pop()
---> 54         self.percDown(1)
     55         return retval
     56 

<ipython-input-1-1dd12c072ab6> in percDown(self, i)
     31             mc = self.minChild(i)
     32             if self.list[i] > self.list[mc]:
---> 33                 tmp = self.list[i]
     34                 self.list[i] = self.list[mc]
     35                 self.list[mc] = tmp

KeyboardInterrupt:
```

In [47]:

```python
times
```

Out[47]:

```
array([1.00350380e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00088120e-03,
       2.99978256e-03, 3.00002098e-03, 7.00163841e-03, 1.89836025e-02,
       3.20072174e-02, 7.10296631e-02, 1.63076878e-01, 3.38948488e-01,
       7.46915102e-01, 1.66279936e+00, 3.68977952e+00, 7.81150126e+00,
       1.69718349e+01, 3.62357903e+01, 7.84339437e+01, 1.68431542e+02,
       3.64554195e+02, 7.74962467e+02, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00])
```

In [48]:

```python
import time
import numpy as np
from random import randint

n=31-2 #szacowany czas ostatniego 5-5.7h
n=31-3 # szacowany czas ostatniego 2.3h
Tim=np.zeros(n+1)
numbers=np.random.randint(-2**n,2**n,2**n)

for i in range(n+1):
    a=BinHeap(numbers[:2**i])
    T=time.time()
    a.sort()
    Tim[i]=time.time()-T
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-48-66a24169f6f1> in <module>
      9 
     10 for i in range(n+1):
---> 11     a=BinHeap(numbers[:2**i])
     12     T=time.time()
     13     a.sort()

<ipython-input-1-1dd12c072ab6> in __init__(self, *items)
      7         self.list = [0]
      8         self.size = 0
----> 9         self.add(*items)
     10 
     11     def percUp(self,i):

<ipython-input-1-1dd12c072ab6> in add(self, *items)
     94             elif isinstance(item, np.ndarray):
     95                 item=list(item)
---> 96                 self.add(item)
     97             else:
     98                 self.insert(item)

<ipython-input-1-1dd12c072ab6> in add(self, *items)
     86         for item in items:
     87             if type(item) in (tuple, list):
---> 88                 self.add(*item)
     89             elif type(item) in (dict, set):
     90                 self.add(*list(item))

<ipython-input-1-1dd12c072ab6> in add(self, *items)
     96                 self.add(item)
     97             else:
---> 98                 self.insert(item)
     99 
    100     def _clear(self): # czyszczenie kopca (prywatna, by nie użyć przez przypadek)

<ipython-input-1-1dd12c072ab6> in insert(self, k)
     20         self.list.append(k)
     21         self.size = self.size + 1
---> 22         self.percUp(self.size)
     23 
     24     def findMin(self):

<ipython-input-1-1dd12c072ab6> in percUp(self, i)
     11     def percUp(self,i):
     12         while i // 2 > 0:
---> 13             if self.list[i] < self.list[i // 2]:
     14                 tmp = self.list[i // 2]
     15                 self.list[i // 2] = self.list[i]

KeyboardInterrupt:
```

In [49]:

```python
Tim
```

Out[49]:

```
array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00040436e-03,
       9.99927521e-04, 3.00073624e-03, 6.00147247e-03, 1.50039196e-02,
       3.60085964e-02, 7.40170479e-02, 1.59030199e-01, 3.48716497e-01,
       7.71439791e-01, 1.65170741e+00, 3.59301019e+00, 7.75503492e+00,
       1.69674728e+01, 3.60844564e+01, 7.80425515e+01, 1.67152969e+02,
       3.60247869e+02, 7.79478208e+02, 1.67020568e+03, 0.00000000e+00,
       0.00000000e+00])
```

In [51]:

```python
import matplotlib.pyplot as plt
```

In [59]:

```python
y=times[:26]
x=range(26)
```

Poniżej jest wykres czasu wykonania funkcji (na czerwono). Jeśli porównamy z wykresem funkcji $n \ln n$ (na niebiesko) to zauważymym, że obie funkcje niemal dokładnie się pokrywają, nasza (czerwona) jest delikatni niżej, ponieważ w zależności od przypadku liczba wykonujemy mniej iteracji (tak jak można było się domyślać). Oczywiście oś niebieska i czerwona się różnią, ale różnica miedzy jedną a drugą to "stała czasowa" (na niebiesko ile zrobiliśmy iteracji, na czerwono jaki czas trwały. czerwony = niebieski * "stała").

In [62]:

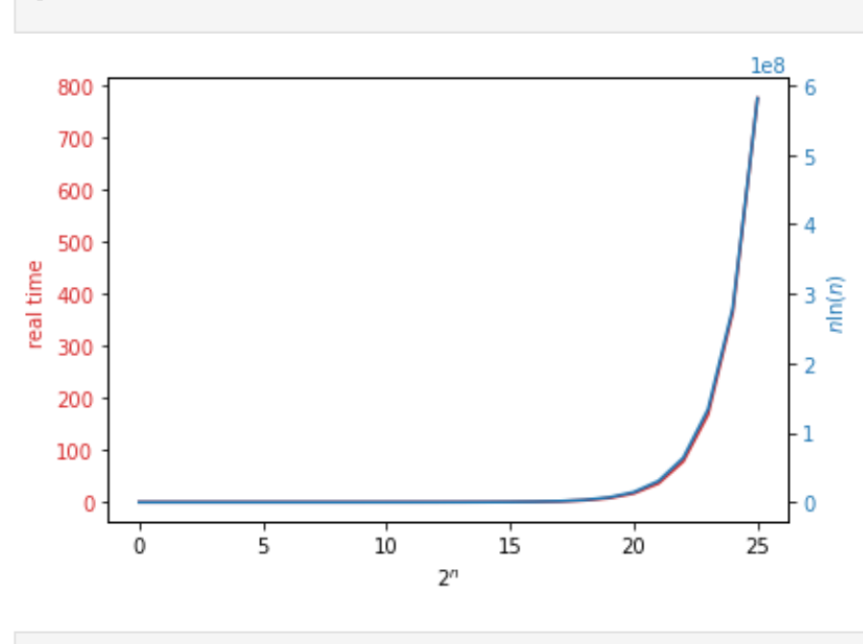
```python
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('$2^n$')
ax1.set_ylabel('real time', color=color)
ax1.plot(x, y, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('$n\ln(n)$', color=color)  # we already handled the x-label with ax1
ax2.plot(x, [2**i*np.log(2**i) for i in x], color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



In [68]:

```python
y=Tim[:-2] # dwa ostatnie nie dały rady się zrobić
x=range(len(Tim[:-2]))
```

In [71]:

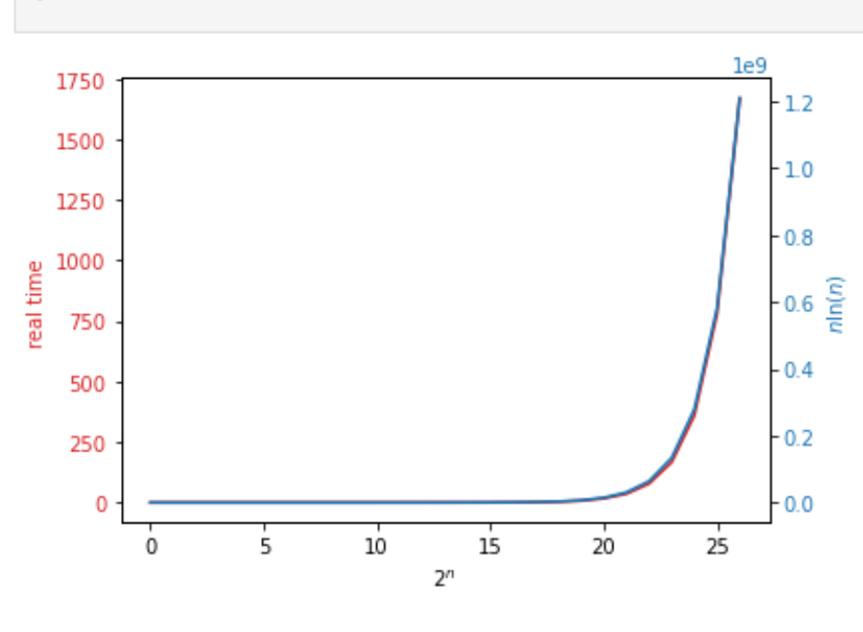
```python
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('$2^n$')
ax1.set_ylabel('real time', color=color)
ax1.plot(x, y, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('$n\ln(n)$', color=color)  # we already handled the x-label with ax1
ax2.plot(x, [2**i*np.log(2**i) for i in x], color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



Z wykresów możemy odczytać, że złożoność obliczeniowe jest rzędu $O(n \ln n)$

Dlaczego tak?

Algorytm ten posiada pesymistyczną klasę złożoności czasowej $O(n \ln n)$ (liniowo logarytmiczną) - każde wstawienie nowego elementu na stos może wymagać tyle przesunięć, ile jest węzłów kopca na ścieżce od miejsca wstawiania do korzenia drzewa. Nie trudno zauważyć, iż liczba ta jest równa lub mniejsza od wysokości kopca, czyli $\log_2 n$. Ponieważ operację tę wykonujemy $n-1$ razy, to dostajemy górne oszacowanie właśnie $O(n \ln n)$.

Zadanie 3

Jest to BinHeap z zmodyfikowanym jedynie konstruktorem oraz metodą insert

In [215…

```python
import types
import numpy as np

class BinHeapN:
    def __init__(self,n:int,*items): # n - maksymalna wielkość kopca
        if type(n) is not int or not n >0:
            raise ValueError("Maksymalna wielkość musi być liczbą naturalną (>0)")
        self.max_size=n
        self.list = [0]
        self.size = 0
        self.add(*items)

    def percUp(self,i):
        while i // 2 > 0:
            if self.list[i] < self.list[i // 2]:
                tmp = self.list[i // 2]
                self.list[i // 2] = self.list[i]
                self.list[i] = tmp
            i = i // 2

    def insert(self,k):
        self.list.append(k)
        self.size = self.size + 1
        if self.size<=self.max_size: # jeśli wielkość jest mniejsza niż maksymalna
            self.percUp(self.size)
        else:
            if k>self.findMin(): # robi coś jeśli dodajemy element większy (mniejszy od razu jest usuwany)
                self.percUp(self.size)
                self.delMin()
            else:
                self.list.pop() # jeśli nowy jest najmniejszy do go usuwamy (dodany na końcu i jeszcze nie prze

    def findMin(self):
        if self.size==0:
            raise IndexError("Kopiec jest pusty")
        return self.list[1]

    def percDown(self,i):
        while (i * 2) <= self.size:
            mc = self.minChild(i)
            if self.list[i] > self.list[mc]:
                tmp = self.list[i]
                self.list[i] = self.list[mc]
                self.list[mc] = tmp
            i = mc

    def minChild(self,i):
        if i * 2 + 1 > self.size:
            return i * 2
        else:
            if self.list[i*2] < self.list[i*2+1]:
                return i * 2
            else:
                return i * 2 + 1

    def delMin(self):
        if self.size==0:
            raise IndexError("Kopiec jest pusty")
        retval = self.list[1]
        self.list[1] = self.list[self.size]
        self.size = self.size - 1
        self.list.pop()
        self.percDown(1)
        return retval

    def buildHeap(self,alist):
        i = len(alist) // 2
        self.size = len(alist)
        self.list = [0] + alist[:]
        while (i > 0):
            self.percDown(i)
            i = i - 1

    def size(self):
        return self.size

    def isEmpty(self):
        return self.size == 0

    def __str__(self):
        txt = "{}".format(self.list[1:])
        return txt

###################################################################
#                             Własne                              #
###################################################################

    "nie zmienione w stosunku do BinHeap"

    def __repr__(self):
        return str(self.list[1:])

    def __len__(self):
        return self.size

    def add(self,*items):
        for item in items:
            if type(item) in (tuple, list):
                self.add(*item)
            elif type(item) in (dict, set):
                self.add(*list(item))
            elif type(item) == types.GeneratorType:
                for i in item:
                    self.add(i)
            elif isinstance(item, np.ndarray):
                item=list(item)
                self.add(item)
            else:
                self.insert(item)

    def _clear(self):
        self.list=[0]
        self.size=0

    def sort(self):
        sortedList=[]
        s=self.size
        while self.size>0:
            sortedList.append(self.delMin())
        self.list=[0]+sortedList
        self.size=s
```

In [216…

```python
a=BinHeapN(5,[2,3,2])
```

In [217…

```python
a
```

Out[217…

```
[2, 3, 2]
```

In [218…

```python
a.add(12,12)
```

In [219…

```python
a
```

Out[219…

```
[2, 3, 2, 12, 12]
```

In [220…

```python
a.add(3)
```

In [221…

```python
a
```

Out[221…

```
[2, 3, 3, 12, 12]
```

In [222…

```python
a.add(5)
```

In [223…

```python
a
```

Out[223…

```
[3, 3, 5, 12, 12]
```

In [224…

```python
a #niezmienione bo 2<3
```

Out[224…

```
[3, 3, 5, 12, 12]
```

Zadanie 4

Funkcja:

- `derivate` pobiera wyrażenie matematyczne i zmienną po której różniczkujemy, łączy wszystkie inne funkcje i zwraca krotkę z drzewem wyrażenia matematycznego (`MathTree`) i jego pochodną również w postaci drzewa.
- `derivate_to_latex` działa podobnie jak poprzednia, ale zwraca jedynie pochodną, ale za to w kodzie $\LaTeX$
- `prepare` zagnieżdża wyrażenia jedno w drugim ("wstawia nawiasy", ustala, że np. mnożenie robimy przed dodawaniem)
- `correct` decyduje o kolejności działań w obrębie działań równoważnych (np. dla $a \cdot b \cdot c$ ustala, które mnożenie pierwsze

In [114…

```python
def derivate(txt, variable): # zwraca drzewko matematyczne
    txt=txt.replace(" ", "")
    txt=txt.replace("**", "^")
    txt=txt.replace("\\", "/")
    txt=txt.replace(":", "/")
    txt=txt.replace("arc", "a")

    tree=MathTreeToTree(*prepare(txt))
    return (tree, MathTreeToTree(tree.derivate(variable)))

def derivate_to_latex(txt,variable):
    txt=txt.replace(" ", "")
    txt=txt.replace("**", "^")
    txt=txt.replace("\\", "/")
    txt=txt.replace(":", "/")
    txt=txt.replace("arc", "a")

    tree=MathTree(*prepare(txt))
    return "$$"+tree.derivate_to_latex(variable)+"$$" # zwraca pochodną w latexu

def correct(equation,lvl=4):
    # lvl - który operator rozpatrujemy (dzielenie przed mnożenim, odejmowanie przed dodawaniem)
    operators=["+","-","*","/","^"]
    if operators[lvl] in ("^",):
        reverse=True
    else:
        reverse=False

    for i in range(len(equation))[::(-1)**reverse]:
        if equation[i] == operators[lvl]:
            return correct(equation[:i-1]+[equation[i-1:i+2]]+equation[i+2:],lvl=lvl)

    if lvl>0:
        return correct(equation,lvl=lvl-1)
    return equation # Jeśli są działania równie ważne (np a*b*c), to wyznacza kolejność działania

def prepare(txt): # Wyznacza kolejność działań itp.
    function=[
        [],
        [],
        ["ln"],
        ["log","pow","exp","sin","cos","tan","cot","sec"],
        ["sqrt","log2","asin","acos","atan","acot","sinh","cosh","tanh","coth","sech"],
        ["log10"],
        ["cosech"]
    ]

    operators=["+","-","*","/","^"]#,"'"]

    unknown=False
    res=[]
    l=len(txt)
    i=0
    while i<l:
        if txt[i]=="(":
            par=0
            j=i
            while True:
                if txt[j]=="(":
                    par+=1

                if txt[j]==")":
                    par-=1

                if par==0:
                    break
                j+=1
            res.append(prepare(txt[i+1:j]))
            i=j
            unnown=True

        elif txt[i:i+6] in function[6]:
            j=i+6
            if txt[j]=="(":
                par=0
                while True:
                    if txt[j]=="(":
                        par+=1

                    if txt[j]==")":
                        par-=1

                    if par==0:
                        break
                    j+=1
                res.append([txt[i:i+6],prepare(txt[i+6+1:j])])
                i=j
            else:
                res.append([txt[i:i+6],txt[i+6]])
            unnown=True

        elif txt[i:i+5] in function[5]:
            #res.append(txt[i:i+5])
            j=i+5
            if txt[j]=="(":
                par=0
                while True:
                    if txt[j]=="(":
                        par+=1

                    if txt[j]==")":
                        par-=1

                    if par==0:
                        break
                    j+=1
                res.append([txt[i:i+5],prepare(txt[i+5+1:j])])
                i=j
            else:
                res.append([txt[i:i+5],txt[i+5]])
            unnown=True

        elif txt[i:i+4] in function[4]:
            j=i+4
            if txt[j]=="(":
                par=0
                while True:
                    if txt[j]=="(":
                        par+=1

                    if txt[j]==")":
                        par-=1

                    if par==0:
                        break
                    j+=1
                res.append([txt[i:i+4],prepare(txt[i+4+1:j])])
                i=j
            else:
                res.append([txt[i:i+4],txt[i+4]])
            unnown=True

        elif txt[i:i+3] in function[3]:
            #res.append(txt[i:i+3])
            j=i+3
            if txt[j]=="(":
                par=0
                while True:
                    if txt[j]=="(":
                        par+=1

                    if txt[j]==")":
                        par-=1

                    if par==0:
                        break
                    j+=1
                res.append([txt[i:i+3],prepare(txt[i+3+1:j])])
                i=j
            else:
                res.append([txt[i:i+3],txt[i+3]])
            unnown=True

        elif txt[i:i+2] in function[2]:
            #res.append(txt[i:i+2])
            j=i+2
            if txt[j]=="(":
                par=0
                while True:
                    if txt[j]=="(":
                        par+=1

                    if txt[j]==")":
                        par-=1

                    if par==0:
                        break
                    j+=1
                res.append([txt[i:i+2],prepare(txt[i+2+1:j])])
                i=j
            else:
                res.append([txt[i:i+2],txt[i+2]])
            unnown=True

        elif txt[i] in operators:
            res.append(txt[i])
            unknown=False

        else:
            if txt[i].isnumeric():
                j=1
                while txt[i:i+j+1].isnumeric() and j<l:
                    j+=1
                if unknown:
                    res.append('*')
                res.append(txt[i:i+j])
                i+=j-1
            else:
                if unknown:
                    res.append("*")
                res.append(txt[i])
            unnown=True



        i+=1
    res=correct(res)

    return res
```

Classa `MathTree`, tu się dzieje cała magia. Pobiera wyrażenie matematyczne przygotowane przez funkcjie `prepare`. Takie wyrażenie jest to lista, która zawiera od 1 do 3 elementów.

- Jeśli zawiera 1, to jest to zmienna, stała, albo liczba, z którą nic więcej nie robimy.
- Przy 2 elementach otrzymujemy funkcję. Pierwszy element to sama funkcja, drugi natomiast to jej argument. Argument jest zapisywany w lewej gałęzi
- Może zawierać jeszcze 3 elementy. Wtedy dostajemy działanie na dwóch wyrażeniach. Pierwszy i trzeci to owe wyrażenia, środkowy to działanie między nimi. (Przykładowo 2^3 daje `['2','^','3']`)

Classa ta zawiera dwie metody

- `__repr__` to wyświetlajnia wyrażenia, oraz zwracania wyniku (w kodzie $\LaTeX$)
- `derivare_to_latex` która zawiera całe mięsko i na niej się skupię

Metoda ta pobiera zmienną po której chcemy różniczkować (jeśli jest ich za dużo, lub za mało zwraca błąd). Lista `function` zawiera listę obsługiwanych funkcji matematycznych, posegregowaną w listach według długości ich nazw. Poniżej jest funkcja różniczkująca. Szuka operacji którą mamy wykonać i zwraca jej pochodną.
Metoda może różniczkować szeroką klasę funkcji (przykładowo $f(x)^{g(x)}$), ale przezto wyniki nie zawsze są w najprostszej postaci (nie skraca ich).
Przykładowo dla

$$f(x) = x^2,$$

funkcjia zwróci wynik, w którym każda działanie oraz liczba będzie w nawiasach, które po opuszczeniu wygląda następująco

$$f'(x) = x^2 \cdot \left(2 \cdot \frac{1}{x} + \ln x \cdot 0\right)$$

Wynik jest poprawny, natomiast nie skrócony (no można się przyczepić co dla $x <= 0$).

Poniżej są dwie klasy, ponieważ jedna zwraca kod $\LaTeX$ a druga nie. Różnią się metodą `derivate` (druga nie ma kodu $\LaTeX$) oraz `__repr__` (to samo + więcej nawiasów).

In [114…

```python
class MathTree:

    def __init__(self, equation):
        #print(1)
        self.key=None
        self.left=None
        self.right=None
        if len(equation)==1 or type(equation)==str:

            while type(equation) is list and len(equation) == 1:
                equation=equation[0]

            if type(equation) is list:
                if len(equation)==2:
                    self.key=equation[0]
                    self.left=MathTree(equation[1])
                else:
                    self.key=equation[1]
                    self.left=MathTree(equation[0])
                    self.right=MathTree(equation[2])
                    #print(type(self.right),type(self.key),type(self.left))
            else:
                self.key=equation

        elif len(equation)==2:
            self.key=equation[0]
            while type(equation[1]) is list and len(equation[1]) == 1:
                equation[1]=equation[1][0]
            self.left=MathTree(equation[1])

        elif len(equation)==3:
            self.key=equation[1]
            self.left=MathTree(equation[0])
            self.right=MathTree(equation[2])

    def __repr__(self):
        if self.right==None:
            if self.left==None:
                return self.key
            else:
                return self.key+r"\left("+str(self.left)+r"\right)"
        else:
            return r"\left("+str(self.left)+r"\right)" + self.key +r"\left("+str(self.right)+r"\right)"



    def derivate_to_latex(self,variable):
        if len(variable)!=1 or variable.isnumeric():
            raise ValueError("Błędna zmienna")

        function=[
            [],
            [],
            ["ln"],
            ["log","pow","exp","sin","cos","tan","cot","sec","csch"],
            ["sqrt","log2","asin","acos","atan","acot","sinh","cosh","tanh","coth","sech"],
            ["log10"],
            []#["cosech"]
        ]

        #print(self.left.key)

        if self.key in ("+","-"):
            return self.left.derivate_to_latex(variable) + self.key + self.right.derivate_to_latex(variable)

        elif self.key=="*":
            return self.left.derivate_to_latex(variable)+r"\cdot "+str(self.right)+r"+"+str(self.left)+r"\cdot 

        elif self.key=="/":
            return r"\dfrac{\left(\left("+self.left.derivate_to_latex(variable)+r"\right)\cdot \left("+str(self

        elif self.key=="^":
            return r"\left("+str(self.left)+r"\right)^{\left("+str(self.right)+r"\right)}\cdot \left(\left("+st

        else:
            if self.key=="ln":
                return r"\dfrac{"+self.left.derivate_to_latex(variable)+r"}{"+str(self.left)+r"}"
            elif self.key=="log":
                return r"\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\left("+str(self.left)+r"\right)\co
            elif self.key=="log2":
                return r"\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\left("+str(self.left)+r"\right)\co
            elif self.key=="exp":
                return r"e^{"+str(self.left)+r"}\cdot \left("+self.left.derivate_to_latex(variable)+r"\right)"
            elif self.key=="sin":
                return r"\cos\left("+str(self.left)+r"\right)\cdot \left("+self.left.derivate_to_latex(variable
            elif self.key=="cos":
                return r"-\sin\left("+str(self.left)+r"\right)\cdot \left("+self.left.derivate_to_latex(variabl
            elif self.key=="tan":
                return r"\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\cos\left("+str(self.left)+r"\right
            elif self.key=="cot":
                return r"-\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\sin\left("+str(self.left)+r"\righ
            elif self.key=="sec":
                return r"\sec\left("+str(self.left)+r"\right)\cdot \tan\left("+str(self.left)+r"\right)\cdot \l
            elif self.key=="csc":
                return r"-\csc\left("+str(self.left)+r"\right)\cdot \cot\left("+str(self.left)+r"\right)\cdot \
            elif self.key=="sqrt":
                return r"\dfrac{"+self.left.derivate_to_latex(variable)+r"}{2\sqrt{"+str(self.left)+r"}}"
            elif self.key=="asin":
                return r"\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\sqrt{1-\left("+str(self.left)+r"\r
            elif self.key=="acos":
                return r"-\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\sqrt{1-\left("+str(self.left)+r"\
            elif self.key=="":
                return r"\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\left("+str(self.left)+r"\right)^2+
            elif self.key=="":
                return r"-\dfrac{"+self.left.derivate_to_latex(variable)+r"}{\left("+str(self.left)+r"\right)^2
            elif self.key=="sinh":
                return r"\cosh\left("+str(self.left)+r"\right)\cdot \left("+self.left.derivate_to_latex(variabl
            elif self.key=="cosh":
                return r"\sinh\left("+str(self.left)+r"\right)\cdot \left("+self.left.derivate_to_latex(variabl
            elif self.key=="tanh":
                return r"\sech\left("+str(self.left)+r"\right)^2\cdot \left("+self.left.derivate_to_latex(varia
            elif self.key=="coth":
                return r"-\csch\left("+str(self.left)+r"\right)^2\cdot \left("+self.left.derivate_to_latex(vari
            elif self.key=="sech":
                return r"-\sech\left("+str(self.left)+r"\right)\cdot \tanh\left("+str(self.left)+r"\right)\cdot
            elif self.key=="":
                return r"-\sech\left("+str(self.left)+r"\right)\cdot \tanh\left("+str(self.left)+r"\right)\cdot
            elif self.key=="csch":
                return r"-\csch\left("+str(self.left)+r"\right)\cdot \coth\left("+str(self.left)+r"\right)\cdot
            elif self.key=="":
                return
            elif self.key=="":
                return

            if self.key==variable:
                return r"1"
            else:
                return r"0"
```

In [114…

```python
class MathTreeToTree:
    def __init__(self, equation):
        self.key=None
        self.left=None
        self.right=None
        if len(equation)==1 or type(equation)==str:

            while type(equation) is list and len(equation) == 1:
                equation=equation[0]

            if type(equation) is list:
                if len(equation)==2:
                    self.key=equation[0]
                    self.left=MathTreeToTree(equation[1])
                else:
                    self.key=equation[1]
                    self.left=MathTreeToTree(equation[0])
                    self.right=MathTreeToTree(equation[2])
            else:
                self.key=equation

        elif len(equation)==2:
            self.key=equation[0]
            while type(equation[1]) is list and len(equation[1]) == 1:
                equation[1]=equation[1][0]
            self.left=MathTreeToTree(equation[1])

        elif len(equation)==3:
            self.key=equation[1]
            self.left=MathTreeToTree(equation[0])
            self.right=MathTreeToTree(equation[2])

    def __repr__(self):
        if self.right==None:
            if self.left==None:
                return self.key
            else:
                return "("+self.key+r"("+str(self.left)+r"))"
        else:
            return r"("+str(self.left)+r")" + self.key +r"("+str(self.right)+r")"


    def derivate(self,variable):
        if len(variable)!=1 or variable.isnumeric():
            raise ValueError("Błędna zmienna")

        function=[
            [],
            [],
            ["ln"],
            ["log","pow","exp","sin","cos","tan","cot","sec","csch"],
            ["sqrt","log2","asin","acos","atan","acot","sinh","cosh","tanh","coth","sech"],
            ["log10"],
            []#["cosech"]
        ]



        if self.key in ("+","-"):
            return "("+self.left.derivate(variable) + self.key + self.right.derivate(variable)+")"

        elif self.key=="*":
            return "("+self.left.derivate(variable)+r"* "+str(self.right)+r"+"+str(self.left)+r"* "+self.right.

        elif self.key=="/":
            return r"((("+self.left.derivate(variable)+r")* ("+str(self.right)+r")) - (("+str(self.left)+r")* (

        elif self.key=="^":
            return r"("+str(self.left)+r")^(("+str(self.right)+r"))* (("+str(self.right)+r")* (("+self.left.der

        else:
            if self.key=="ln":
                return r"("+self.left.derivate(variable)+r")/("+str(self.left)+r")"
            elif self.key=="log":
                return r"("+self.left.derivate(variable)+r")/(("+str(self.left)+r")* ln (10))"
            elif self.key=="log2":
                return r"("+self.left.derivate(variable)+r")/(("+str(self.left)+r")* ln(2))"
            elif self.key=="exp":
                return r"exp("+str(self.left)+r")* ("+self.left.derivate(variable)+r")"
            elif self.key=="sin":
                return r"cos("+str(self.left)+r")* ("+self.left.derivate(variable)+r")"
            elif self.key=="cos":
                return r"-sin("+str(self.left)+r")* ("+self.left.derivate(variable)+r")"
            elif self.key=="tan":
                return r"("+self.left.derivate(variable)+r")/(\cos("+str(self.left)+r"))"
            elif self.key=="cot":
                return r"-("+self.left.derivate(variable)+r")/(\sin("+str(self.left)+r"))"
            elif self.key=="sec":
                return r"sec("+str(self.left)+r")* tan("+str(self.left)+r")* ("+self.left.derivate(variable)+r"
            elif self.key=="csc":
                return r"-csc("+str(self.left)+r")* cot("+str(self.left)+r")* ("+self.left.derivate(variable)+r
            elif self.key=="sqrt":
                return r"("+self.left.derivate(variable)+r")/(sqrt("+str(self.left)+r"))"
            elif self.key=="asin":
                return r"("+self.left.derivate(variable)+r")/(sqrt(1-("+str(self.left)+r")^2))"
            elif self.key=="acos":
                return r"-("+self.left.derivate(variable)+r")/(sqrt(1-("+str(self.left)+r")^2))"
            elif self.key=="":
                return r"("+self.left.derivate(variable)+r")/(("+str(self.left)+r")^2+1)"
            elif self.key=="":
                return r"-("+self.left.derivate(variable)+r")/(("+str(self.left)+r")^2+1)"
            elif self.key=="sinh":
                return r"cosh("+str(self.left)+r")* ("+self.left.derivate(variable)+r")"
            elif self.key=="cosh":
                return r"sinh("+str(self.left)+r")* ("+self.left.derivate(variable)+r")"
            elif self.key=="tanh":
                return r"sech("+str(self.left)+r")^2* ("+self.left.derivate(variable)+r")"
            elif self.key=="coth":
                return r"-csch("+str(self.left)+r")^2* ("+self.left.derivate(variable)+r")"
            elif self.key=="sech":
                return r"-sech("+str(self.left)+r")* tanh("+str(self.left)+r")* ("+self.left.derivate(variable)
            elif self.key=="":
                return r"-sech("+str(self.left)+r")* tanh("+str(self.left)+r")* ("+self.left.derivate(variable)
            elif self.key=="csch":
                return r"-csch("+str(self.left)+r")* coth("+str(self.left)+r")* ("+self.left.derivate(variable)
            elif self.key=="":
                return
            elif self.key=="":
                return

            if self.key==variable:
                return r"1"
            else:
                return r"0"
```

Będę korzystał z klasy `MathTree`, ale zanim zacznę to pokarzę, że klasa druga też działa, na prostym przykładzie.

In [115…

```python
a="x^2"
```

In [115…

```python
a
```

Out[115…

```
'x^2'
```

In [115…

```python
a=derivate("x^2","x")[1]
```

In [115…

```python
type(a)
```

Out[115…

```
__main__.MathTreeToTree
```

In [115…

```python
a
```

Out[115…

```
(x)^((2))* ((2)* ((1))/((x))+ln(x)* (0))
```

Dostaliśmy pochodną wyrażenia (`[1]` by pierwszy element to drzewo wyrażenia, a dopiero drugi to jego pochodna). Zróżniczkujmy ją jeszcze raz!

In [115…

```python
a=derivate(str(a),'x')[1]
```

In [115…

```python
type(a)
```

Out[115…

```
__main__.MathTreeToTree
```

In [115…

```python
a
```

Out[115…

```
((x)^((2))* ((2)* ((1))/((x))+ln(x)* (0))* ((2)*((1)/(x)))+(((ln(x)))*(0))+(x)^(2)* ((0* (1)/(x)+2* (((0)* (x))
- ((1)* (1)))/((x)^2))+((1)/(x)* 0+(ln(x))* 0)))
```

Wyrażenie jest skomplikowane, ale jeśli je uprościmy to otrzymamy `2` co jest poprawnym wynikiem, ponieważ

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(x^2\right) = 2$$

Przy okazji mogę polecić stronę https://www.derivative-calculator.net/, nie tylko do liczenia pochodnych (co się to bardzo przydało), ale również do skracania wyniku. Po wkeljenu powyższego wyniku wystarczy dać oblicz (`Go!`) i zamin policzy to uprości wynik.

W dalszej części urzywam pierwszej klasy, by uprościć wyniki (by dało się coś z tego rozszyfrować).

W przykładach kożystałem z rozszerzenia jupytera `Python Markdown`, które pozwala uzywać zmiennych jupytera (pythona) w markdown'ie. Przykładowo

In [116…

```python
a=derivate_to_latex("x^2","x")
print(a)
```

```
$$\left(x\right)^{\left(2\right)}\cdot \left(\left(2\right)\cdot \dfrac{\left(1\right)}{\left(x\right)}+\ln\lef
t(x\right)\cdot \left(0\right)\right)$$
```

Zamiast przekopiować powyższy kod wystarczy dać `{ {a} }` (użyłem teraz spacji by rozszerzenie nie działało), by otrzymać poniższy wynik

{{a}}

Uwaga! By dane rozszerzenie działało notatnik musi być zaufany, oraz niestety nie ma możliwości wyeksportować pliku z poniższymi wynikami.

Przykłady innych pochodnych

In [116…

```python
a=derivate_to_latex("cos(x)^(sin(x)+x^2)","x")
```

{{a}}

In [116…

```python
a=derivate_to_latex("ax^2+bx+c-12","x")
```

Działą z parametrami {{a}}

In [116…

```python
a=derivate_to_latex("ax^2+bx+c-12","a")
```

Nieważne jakie wybierzemy (przykładowo a) {{a}}

In [116…

```python
a=derivate_to_latex("ax^2+bx+c-12","1")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1167-f065a5f7f65e> in <module>
----> 1 a=derivate_to_latex("ax^2+bx+c-12","1")

<ipython-input-1142-99c237e9d0e3> in derivate_to_latex(txt, variable)
     17
     18     tree=MathTree(*prepare(txt))
---> 19     return "$$"+tree.derivate_to_latex(variable)+"$$" # zwraca pochodną w latexu
     20
     21 def correct(equation,lvl=4):

<ipython-input-1144-28db9166b27c> in derivate_to_latex(self, variable)
     47     def derivate_to_latex(self,variable):
     48         if len(variable)!=1 or variable.isnumeric():
---> 49             raise ValueError("Błędna zmienna")
     50
     51         function=[

ValueError: Błędna zmienna
```

Po liczbie nie działa

In [116…

```python
a=derivate_to_latex("23x","x")
```

Można ominąć znaki mnożeni {{a}}

Oraz dać więcej złożeń funkcji (gdy będzie za dużo jupyter ma problem z przetworzeniem kodu $\LaTeX$, ale wynik pozostaje poprawny)

In [117…

```python
a=derivate_to_latex("sin(x)^cos(x)^x",'x')
```

{{a}}

A i oczywiście ułamki też działają

In [117…

```python
a=derivate_to_latex("x^(1/3)",'x')
```

{{a}}