Potrzebne importy

In [1]:

```
import os
import time
from queue import PriorityQueue
from bitarray import bitarray
```

Zadanie Kodowanie Huffmana statyczne

In [46]:

```
class Node:
    def __init__(self, weight = 0, symbol= None,  parent= None, left= None, right= None)
:
        self.w = weight
        self.symbol = symbol
        self.parent = parent
        self.left = left
        self.right = right

    def Dictionary(self, dictionary, sipher = bitarray()):
        if self.symbol:
            dictionary[self.symbol] = sipher

        if self.left:
            other = sipher.copy()
            other.append(0)
            self.left.Dictionary(dictionary, other)

        if self.right:
            other = sipher.copy()
            other.append(1)
            self.right.Dictionary(dictionary, other)

    def __lt__(self, other):
        return self.w < other.w

    def __le__(self, other):
        return self.w <= other.w
```

In [48]:

```
class HuffmanTree:
    def __init__(self, freq):
        q = PriorityQueue()
        for l,w in freq.items():
            q.put((w,Node(weight = w,symbol = l)))

        while q.qsize() > 1:
            first = q.get()[1]
            second = q.get()[1]
            other = Node(weight = first.w+second.w, left=first, right=second)
            q.put((other.w,other))

        self.root = q.get()[1]
        self.codes = {}
        self.root.Dictionary(self.codes)

    @property
    def codeDict(self):
        if self.root.left == self.root.right:
            code = bitarray()
            code.append(0)
```

```
            self.root.symbol = code
            return

        return self.codes

    def encode(text):
        unique = {}
        table = bitarray()
        m = bitarray()

        for el in text:
            if el in unique.keys():
                unique[el] += 1
            else:
                unique[el] = 1

        tree = HuffmanTree(unique)
        for el, freq in unique.items():
            symbol = bitarray()
            repeated = bitarray()
            symbol.frombytes(el.encode('utf-8'))
            repeated.frombytes(freq.to_bytes(4, byteorder='big', signed=False))
            table += symbol
            table += repeated

        n = len(table)
        m.frombytes(n.to_bytes(4, byteorder='big', signed=False))
        bites = m + table
        for el in text:
            bites += tree.codeDict[el]

        return bites

    def decode(encrypted):
        n = int.from_bytes(encrypted[:32], byteorder='big', signed=True)
        res = ""
        signs = {}
        bites = encrypted[32:]
        for i in range(0,n,40):
            freq = 0
            first = bites[:8]
            second = bites[8:]
            bites = second
            letter = first.tobytes().decode()
            first = bites[:32]
            second = bites[32:]
            for bit in first:
                freq = (freq << 1) | bit

            bites = second
            signs[letter] = freq

        tree = HuffmanTree(signs)
        n = len(bites)
        for i in range(n):
            leaf = tree.root
            while leaf.left != leaf.right:
                if not bites[i]:
                    leaf = leaf.left
                else:
                    leaf = leaf.right

            res += leaf.symbol
```

In [50]:

```
print("1kb file")
print("Ratio : " + str(getTimes("1kB.txt")) + "%")
print("-------------------------------------------")
print("10kb file")
print("Ratio : " + str(getTimes("10kB.txt")) + "%")
```

```
print("-----------------------------------------------")
print("100kb file")
print("Ratio : " + str(getTimes("100kB.txt")) + "%")
print("-----------------------------------------------")
print("1mb file")
print("Ratio : " + str(getTimes("1mB.txt")) + "%")
```

```
1kb file
Coding : 0.0010001659393310547 s
Decoding : 0.006999015808105469 s
Ratio : 84.50413223140497%
-----------------------------------------------
10kb file
Coding : 0.00799703598022461 s
Decoding : 0.1249990463256836 s
Ratio : 55.37442153975599%
-----------------------------------------------
100kb file
Coding : 0.08100056648254395 s
Decoding : 1.294999122619629 s
Ratio : 53.474261107848896%
-----------------------------------------------
1mb file
Coding : 0.7869980335235596 s
Decoding : 13.729997873306274 s
Ratio : 53.28786331634722%
```

In [ ]:

```
```