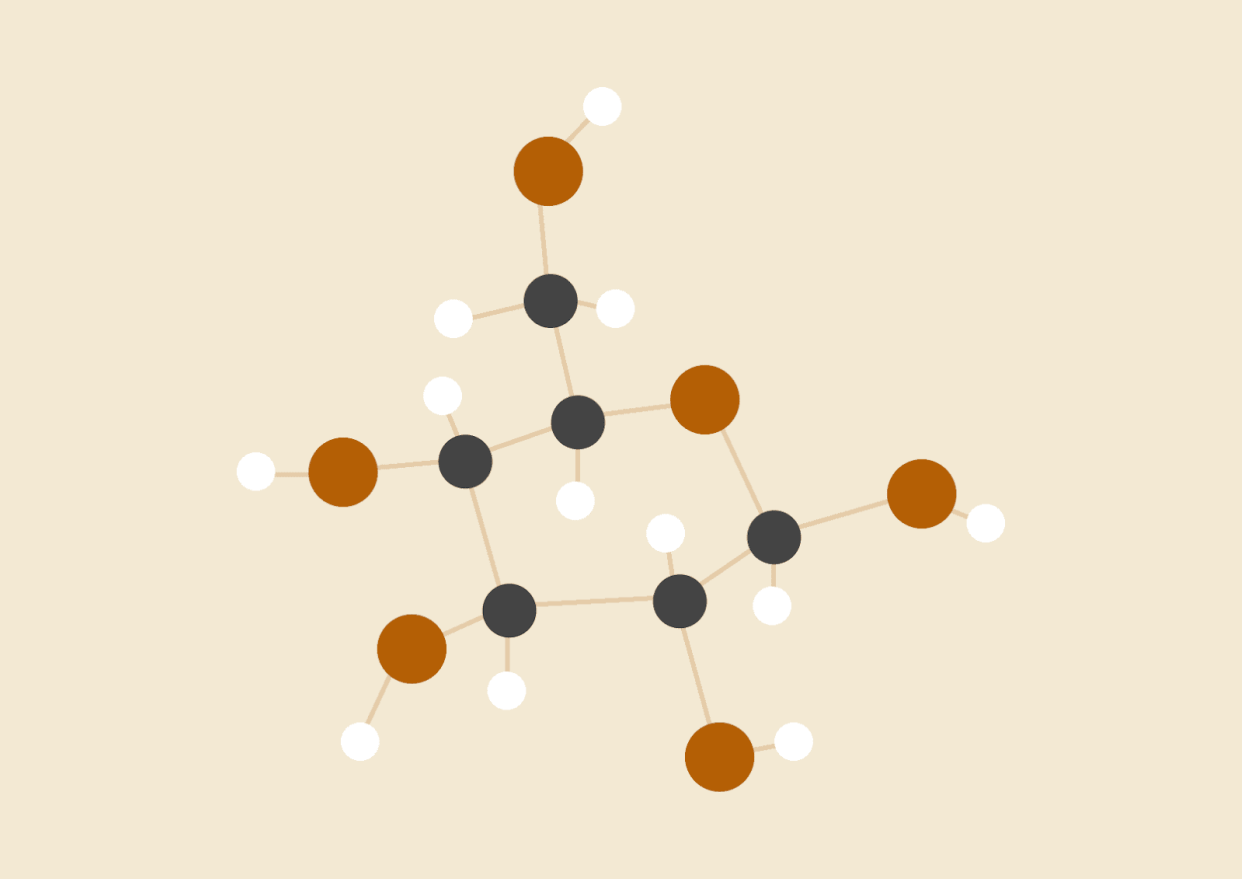
ЛАБОРАТОРНАЯ РАБОТА

Алгоритм Хаффмана



Боглачев А. С. (R3142)
Дженжеруха К.А. (R3142)
Юманов М. А. (R3137)

04.09.2022
Дискретная Математика

ЦЕЛЬ

Реализовать алгоритм Хаффмана на языке программирования Phyton 3.

ЗАДАЧИ

1.Реализовать кодирование символов в бинарный код с помощью алгоритма Хаффмана.

2.Реализовать декодирование бинарного кода в символы с помощью алгоритма Хаффмана

Ход работы

1) Объявляем переменные и открываем файл

```
operation, input_name, output_name = map(str, input().split())
with open(input_name) as f:
    text = f.readlines()
```

2) Объявляем класс вершин и в нем прописываем 2 функции

```
class Node:
    ch = ''
    freq = 0
    left = None
    right = None

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

    def __init__(self, ch, freq, left, right):
        self.ch = ch
        self.freq = freq
        self.left = left
        self.right = right
```

3) определяем какую операцию будем выполнять и записываем результат в файл

```
if operation == "--encode":
    encoding_text = reduce(lambda x, y: x + y, text)
    dictionary, binary = huffman_tree_encode(encoding_text)
    f = open(output_name, "w")
    f.write(dictionary)
    f.write(binary)
    f.close()
elif operation == "--decode":
    actual_text = huffman_tree_decode(text)
    f = open(output_name, "w")
    f.write(actual_text)
```

4)Определяем частоту символов, записываем в словарь и возвращаем, кодируем символы

```
def huffman_tree_encode(text):
    chars = {}
    for i in range(len(text)):
        ch = text[i]
        if ch in chars.keys():
            chars[ch] += 1
        else:
            chars[ch] = 1

    alphabet = huffman_tree_builder(chars)
    code = encode(alphabet, text)
    dictionary = str(len(chars)) + "\n"
    for ch in chars:
        if str(ch) != "\n":
            dictionary = dictionary + str(ch) + ": " + str(chars[ch]) + "\n"
        else:
            dictionary = dictionary + "\\n" + ": " + str(chars[ch]) + "\n"

    # code = map(int, code)
    # binary = bytes(code)
    # print(code)
    # print(decode(code, root))
    return dictionary, code
```

5)Строим дерево Хаффмана

```
def huffman_tree_builder(chars):
    chars = dict(sorted(chars.items(), key=lambda item: item[1]))
    queue = []
    for ch in chars:
        queue.append(Node(ch, chars[ch], None, None))

    while len(queue) != 1:
        left = queue[0]
        right = queue[1]

        queue.remove(queue[0])
        queue.remove(queue[0])

        freq_sum = right.freq + left.freq
        queue.append(Node('', freq_sum, left, right))

        queue.sort()

    root = queue[0]
    alphabet = {}
    for ch in chars.keys():
        alphabet[ch] = ""
    hufman_tree(root, "", alphabet)
    return alphabet
```

6)Проходим до листа

```
def hufman_tree(root, str, alphabet):
    if root.ch != "":
        alphabet[root.ch] = str
        return

    hufman_tree(root.right, str + '1', alphabet)
    hufman_tree(root.left, str + '0', alphabet)
```

7)Декодируем

```
def huffman_tree_decode(text):
    length = int(text[0])
    chars = {}
    for i in range(1, length+1):
        string = text[i]
        if string[0] == ":":
            chars[":"] = int(string[3])
        else:
            chars[string.split(":")[0]] = int(string.split(":")[1])
    code = text[length + 1]
    alphabet = huffman_tree_builder(chars)
    char_code = ""
    actual_text = ""
    for i in code:
        char_code += i
        if char_code in alphabet.values():
            if list(alphabet.keys())[list(alphabet.values()).index(char_code)] == "\\n":
                actual_text += "\n"
            else:
                actual_text += list(alphabet.keys())[list(alphabet.values()).index(char_code)]
            char_code = ""
    return actual_text
```

РЕЗУЛЬТАТЫ

1. Реализовали кодирование символов в бинарный код с помощью алгоритма Хаффмана.
2. Реализовали декодирование бинарного кода в символы с помощью алгоритма Хаффмана

ВЫВОДЫ

Реализовали алгоритм Хаффмана на языке программирования Phyton 3.