NI-VSM-HW1

Members: Andrei Shchapaniak (shchaand), Maksim Spiridonov (spirimak)

Preparation.

```
'''
# Representative member: Andrei Shchapaniak, 14.05.2002
'''
K = 14
L = len('Shchapaniak')
X = ((K*L*23) % 20) + 1
Y = ((X + ((K*5 + L*7) % 19)) % 20) + 1
'''
file1 = 003.txt
file2 = 018.txt
'''
```

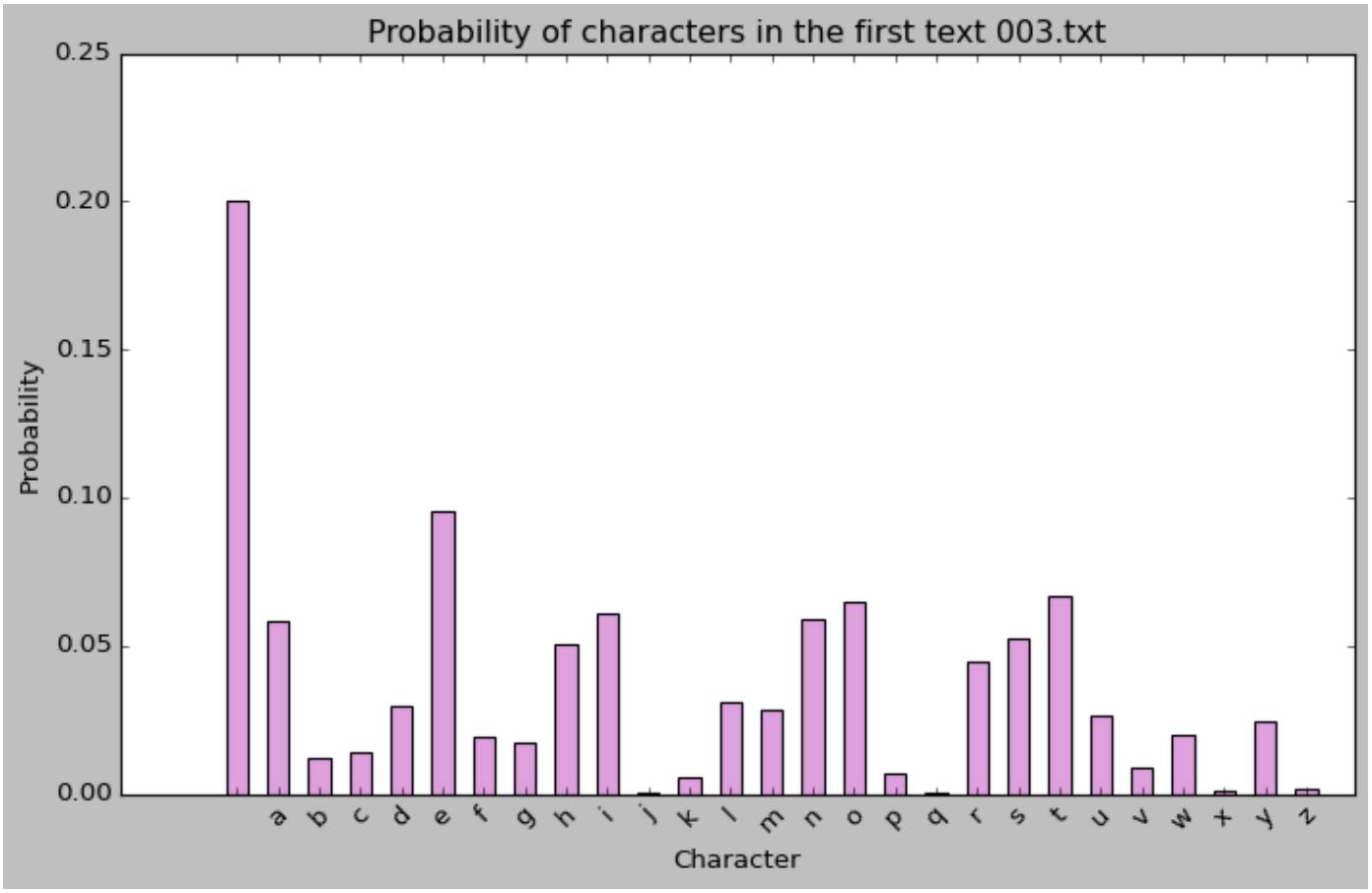
1. Load the texts for analysis from both data files. For each text separately, estimate the probabilities of characters (symbols including space) that occur in the texts. Graphically represent the resulting probabilities.

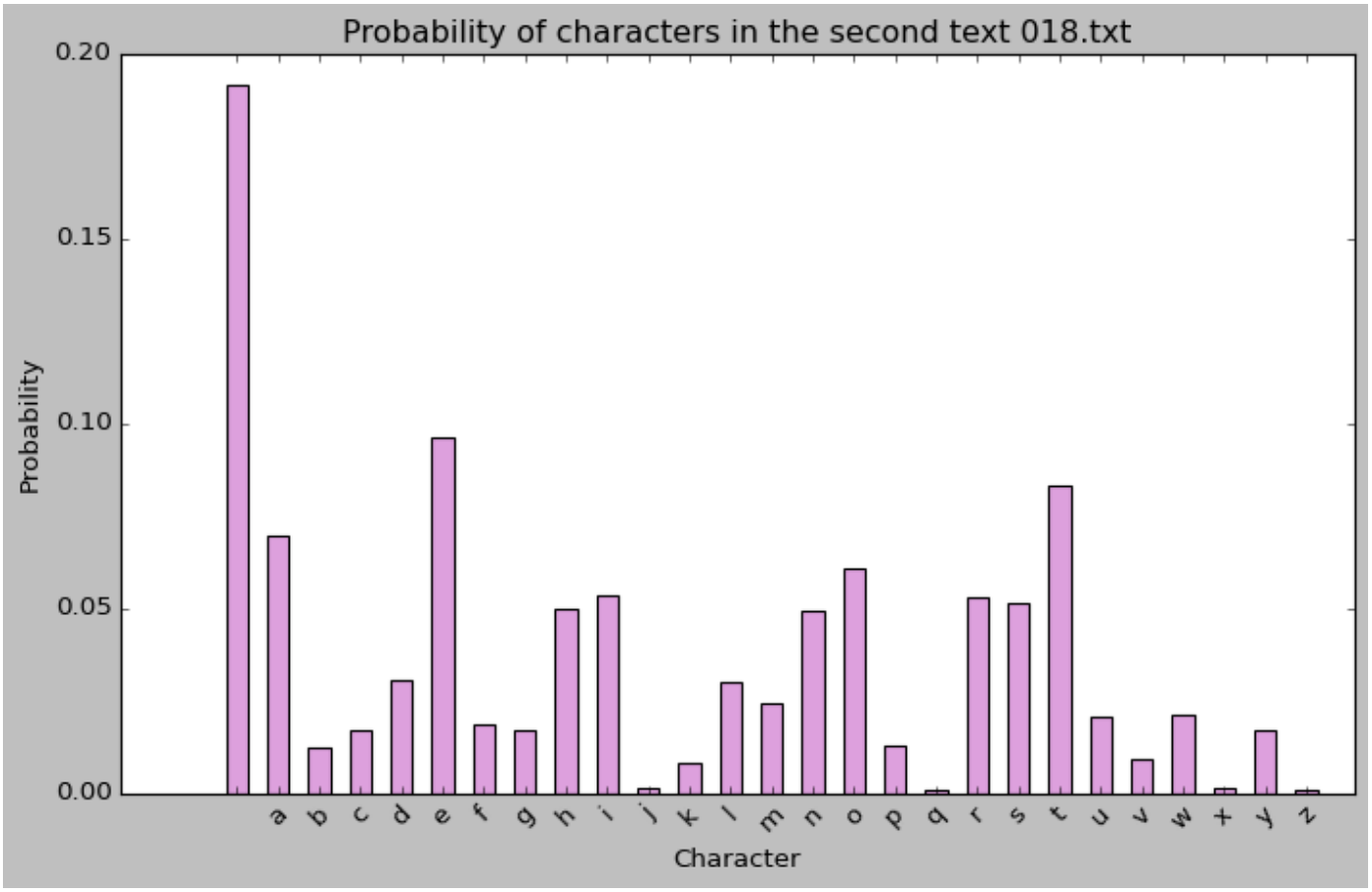
```
import numpy as np
import os

class MyFile:
  def __init__(self, filename):
    self.filename = filename
    self.text = ""
    self.letters = []
    self.letters_pbt = []

  def read_text(self):
      with open(os.path.join('files', self.filename), 'r') as f:
          self.text = f.readlines()[1].strip()

  def calc_letters_pbt(self):
      self.letters, letters_counts = np.unique(list(self.text),
return_counts=True)
      self.letters_pbt = letters_counts / np.sum(letters_counts)
```





---

2. **`For each text separately, calculate the entropy of the estimated character distribution.`**

```
from scipy import stats
import numpy as np

def calc_entropy(letters_pbt, base = 2):
    return stats.entropy(letters_pbt) / np.log(base)
```

The entropy `H(x)` of the estimated character distribution can be calculated using the formula:

$$H(x) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Where `p(x)` is the probability of character `x` in the distribution.

- **Results**

	003.txt	018.txt
Entropy	4.06706492699646	4.08257097934126

---

3. `Find the optimal binary instant code for encoding the characters of the first text. Properly explain why the code you found is optimal!`

```
import heapq

class Huffman:
  def __init__(self, char_pbt):
    self.char_pbt = char_pbt # {'ch': pbt, ...}
    self.huffman_code = []

  def create_code(self):
    heap = [[pbt, [ch, ""]] for ch, pbt in self.char_pbt.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
      child1 = heapq.heappop(heap)
      child2 = heapq.heappop(heap)

      for pair in child1[1:]:
        pair[1] = '0' + pair[1]
      for pair in child2[1:]:
        pair[1] = '1' + pair[1]

      heapq.heappush(heap, [child1[0] + child2[0]] + child1[1:] +
child2[1:])

    self.huffman_code = sorted(heapq.heappop(heap)[1:], key=lambda p:
```

```
(len(p[-1]), p))

  def calc_average_length(self):
    return sum(len(code) * self.char_pbt[ch] for ch, code in
self.huffman_code)
```

- `create_code() function explanation:`
  1. Initialize a priority queue (min-heap) with characters and their probabilities.
  2. Repeatedly remove the two nodes with the lowest probabilities from the queue, combine them into a new node (summing their probabilities), and insert this new node back into the queue.
  3. Assign binary digits ('0' and '1') to the edges connecting nodes to build the Huffman codes for each character.
  4. Repeat untill only one node remains and then extract the Huffman code.
- **Huffman code for the file 003.txt:**

Character	Probability	Representation
' '	0.20035391270153363	"00"
'a'	0.0581989775855289	"0110"
'b'	0.01219032638615808	"1101111"
'c'	0.013763271726307511	"011110"
'd'	0.029689343295320487	"10010"
'e'	0.0951631930790405	"1110"
'f'	0.019071962249311836	"110101"
'g'	0.017499016909162408	"110100"
'h'	0.05072748721981911	"0100"
'i'	0.06075501376327173	"1010"
'j'	0.0005898545025560362	"11011100101"
'k'	0.005701926858041683	"11011101"
'l'	0.03067243413291388	"10011"
'm'	0.028509634290208415	"01110"
'n'	0.05878883208808494	"1000"
'o'	0.0646873771136453	"1011"
'p'	0.007078254030672434	"0111110"
'q'	0.00039323633503735744	"11011100100"
'r'	0.04463232402674007	"11110"

Character	Probability	Representation
's'	0.0526936688950059	"0101"
't'	0.06685017695635077	"1100"
'u'	0.02634683444750295	"111111"
'v'	0.008847817538340543	"0111111"
'w'	0.019661816751867872	"110110"
'x'	0.0011797090051120724	"1101110011"
'y'	0.024184034604797483	"111110"
'z'	0.0017695635076681085	"110111000"

Huffman code is optimal, because it satisfies the necessary condition for optimality, which is given by:

$$H(X) \leq L(C) < H(X) + 1$$

where average length of code `L(C)` is defined as follows:

$$L(C) = \sum_{x \in X} l(x)p(x)$$

- Let's calculate and prove the optimality of the found code for the first file `003.txt`:

Entropy	4.06706492699646
Average length of code	4.103814392449863

$$4.06706492699646 \leq 4.103814392449863 < 5.06706492699646$$

---

4. `For each text separately, calculate the average length of the code C and compare it with the entropy of the character distribution. Is the code C also optimal for the second text? Properly justify!`

- Based on the formulas from Task 3, the results are as follows:

	003.txt	018.txt
Entropy	4.06706492699646	4.08257097934126
Average length of code	4.103814392449863	4.13106020245072

It is clear that both codes for the `003.txt` and `018.txt` texts satisfy the condition of optimality that was discussed in Task 3.