Дискретні структури ІПЗ-21-2

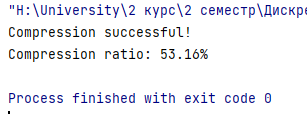
Губарєв Ростислав Вадимович

**Лабораторна робота №9.2**

Створіть програму, яка будує коди Хаффмана та оцінює ступінь стиснення з їхньою допомогою для довільного заданого файлу. \* Створіть програму, яка стискає файли за допомогою кодів Хаффмана.

|  |
| --- |
| **Код**  import heapq import os   class HuffmanNode:  *"""  Представляє вузол дерева Хаффмана.  """* def \_\_init\_\_(self, char, freq):  *"""  Ініціалізує новий об'єкт HuffmanNode.   Параметри:  char (str): Символ, пов'язаний з вузлом.  freq (int): Частота входження символу.  """* self.char = char  self.freq = freq  self.left = None  self.right = None   def \_\_lt\_\_(self, other):  *"""  Порівнює два вузли Хаффмана за їхніми частотами.   Параметри:  other (HuffmanNode): Інший вузол Хаффмана для порівняння.   Повертає:  bool: True, якщо поточний вузол має меншу частоту, ніж інший вузол.  """* return self.freq < other.freq   class HuffmanCoding:  *"""  Клас для створення кодів Хаффмана та стиснення даних.  """* def \_\_init\_\_(self):  self.heap = []  self.codes = {}  self.reverse\_mapping = {}  self.root = None   def build\_frequency\_dict(self, data):  *"""  Побудова словника частот символів у вхідних даних.   Параметри:  data (str): Вхідні дані.   Повертає:  dict: Словник з символами та їх частотами вхідних даних.  """* frequency\_dict = {}  for char in data:  frequency\_dict[char] = frequency\_dict.get(char, 0) + 1  return frequency\_dict   def build\_heap(self, frequency\_dict):  *"""  Побудова купи з вузлів Хаффмана на основі словника частот.   Параметри:  frequency\_dict (dict): Словник з символами та їх частотами.   Повертає:  list: Купа вузлів Хаффмана.  """* heap = []  for char, freq in frequency\_dict.items():  heapq.heappush(heap, HuffmanNode(char, freq))  return heap   def merge\_nodes(self, heap):  *"""  Об'єднання вузлів Хаффмана у дерево.   Параметри:  heap (list): Купа вузлів Хаффмана.   Повертає:  HuffmanNode: Кореневий вузол дерева Хаффмана.  """* while len(heap) > 1:  node1 = heapq.heappop(heap)  node2 = heapq.heappop(heap)  merged = HuffmanNode(None, node1.freq + node2.freq)  merged.left = node1  merged.right = node2  heapq.heappush(heap, merged)  return heap[0]   def build\_codes\_helper(self, root, current\_code):  *"""  Допоміжна функція для побудови кодів Хаффмана.   Параметри:  root (HuffmanNode): Поточний вузол дерева Хаффмана.  current\_code (str): Поточний код символу.  """* if root is None:  return  if root.char is not None:  self.codes[root.char] = current\_code  self.build\_codes\_helper(root.left, current\_code + "0")  self.build\_codes\_helper(root.right, current\_code + "1")   def build\_codes(self, root):  *"""  Побудова кодів Хаффмана на основі дерева Хаффмана.   Параметри:  root (HuffmanNode): Кореневий вузол дерева Хаффмана.  """* self.build\_codes\_helper(root, "")   def get\_encoded\_data(self, data):  *"""  Отримання закодованих даних на основі кодів Хаффмана.   Параметри:  data (str): Вхідні дані.   Повертає:  str: Закодовані дані.  """* encoded\_data = ""  for char in data:  encoded\_data += self.codes[char]  return encoded\_data   def pad\_encoded\_data(self, encoded\_data):  *"""  Додавання доповнення до закодованих даних для вирівнювання.   Параметри:  encoded\_data (str): Закодовані дані.   Повертає:  str: Закодовані дані з доданим доповненням.  """* extra\_padding = 8 - len(encoded\_data) % 8  for i in range(extra\_padding):  encoded\_data += "0"  padded\_info = "{0:08b}".format(extra\_padding)  encoded\_data = padded\_info + encoded\_data  return encoded\_data   def get\_byte\_array(self, padded\_encoded\_data):  *"""  Отримання байтового масиву з доповнених закодованих даних.   Параметри:  padded\_encoded\_data (str): Закодовані дані з доданим доповненням.   Повертає:  bytearray: Байтовий масив з закодованими даними.  """* if len(padded\_encoded\_data) % 8 != 0:  print("Encoded data is not padded properly.")  exit(0)  b = bytearray()  for i in range(0, len(padded\_encoded\_data), 8):  byte = padded\_encoded\_data[i:i + 8]  b.append(int(byte, 2))  return b   def compress(self, input\_file, output\_file):  *"""  Стиснення файлу за допомогою кодів Хаффмана.   Параметри:  input\_file (str): Шлях до вхідного файлу.  output\_file (str): Шлях до стисненого файлу.   Повертає:  str: Шлях до стисненого файлу.  """* with open(input\_file, 'r') as file:  data = file.read()  data = data.rstrip()   frequency\_dict = self.build\_frequency\_dict(data)  heap = self.build\_heap(frequency\_dict)  root = self.merge\_nodes(heap)  self.build\_codes(root)  encoded\_data = self.get\_encoded\_data(data)  padded\_encoded\_data = self.pad\_encoded\_data(encoded\_data)  byte\_array = self.get\_byte\_array(padded\_encoded\_data)   with open(output\_file, 'wb') as file:  file.write(bytes(byte\_array))   print("Compression successful!")  return output\_file   def remove\_padding(self, padded\_encoded\_data):  *"""  Видалення доповнення з закодованих даних.   Параметри:  padded\_encoded\_data (str): Закодовані дані з доданим доповненням.   Повертає:  str: Закодовані дані без доповнення.  """* padded\_info = padded\_encoded\_data[:8]  extra\_padding = int(padded\_info, 2)  padded\_encoded\_data = padded\_encoded\_data[8:]  encoded\_data = padded\_encoded\_data[:-1 \* extra\_padding]  return encoded\_data   def decode\_data(self, encoded\_data, root):  *"""  Розкодування закодованих даних на основі дерева Хаффмана.   Параметри:  encoded\_data (str): Закодовані дані.  root (HuffmanNode): Кореневий вузол дерева Хаффмана.   Повертає:  str: Розкодовані дані.  """* current\_node = root  decoded\_data = ""  for bit in encoded\_data:  if current\_node is None:  break  if bit == '0':  current\_node = current\_node.left  else:  current\_node = current\_node.right   if current\_node is not None and current\_node.char is not None:  decoded\_data += current\_node.char  current\_node = root   return decoded\_data  def calculate\_compression\_ratio(original\_file, compressed\_file):  *"""  Обчислення ступеня стиснення між оригінальним файлом та стисненим файлом.   Параметри:  original\_file (str): Шлях до оригінального файлу.  compressed\_file (str): Шлях до стисненого файлу.   Повертає:  float: Ступінь стиснення у відсотках.  """* original\_size = os.path.getsize(original\_file)  compressed\_size = os.path.getsize(compressed\_file)  compression\_ratio = (compressed\_size / original\_size) \* 100  return compression\_ratio   *# Застосовуємо кодування Хаффмана до заданого файлу* input\_file = "input.txt" output\_file = "compressed.bin"  huffman = HuffmanCoding() compressed\_file = huffman.compress(input\_file, output\_file)  *# Розраховуємо ступінь стиснення* compression\_ratio = calculate\_compression\_ratio(input\_file, compressed\_file) print("Compression ratio: {:.2f}%".format(compression\_ratio)) |

**Результат**

****