МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №11 НА ТЕМУ:**

**Сжатие/распаковка данных арифметическим методом**

Ф.И.О.

Божко Денис Владимирович

Преподаватель

асс. Берников Владислав Олегович

Минск 2021

**Цель:** приобретение практических навыков использования арифметических методов сжатия/распаковки данных.

**Теоретические сведения**

Как мы установили, вероятностные (префиксные) методы являются достаточно простыми и эффективными: они основаны на использовании кодов переменной длины и для вероятностей появления символов алфавита, кратных степеням числа 2 (1/2, 1/4, 1/8 и т. п.), дают наилучшие результаты. При других значениях вероятностей, как правило, самый короткий код получается большим, чем двоичный логарифм этой вероятности (взятый с отрицательным знаком). Например, при р(аi) = 0,4 получим −log20,15 = 1,32. Понятно, что мы не можем закодировать этот символ только 2 битами (либо одним), т. е. решение не всегда является оптимальным. Анализируемого недостатка лишены арифметические методы.

Пpи аpифметическом сжатии (кодиpовании) текст пpедставляется вещественными числами в интеpвале от 0 до 1. По меpе анализа текста отобpажающий его интеpвал уменьшается, а количество битов для его пpедставления возpастает. Очеpедные символы текста сокpащают величину интеpвала, исходя из значений соответствующих веpоятностей.

Основная идея арифметического метода сжатия заключается в том, чтобы присваивать коды не отдельным символам, а их последовательностям.

Таким образом, как и во всех энтропийных алгоритмах, исходной является информация о частоте встречаемости каждого символа алфавита.

Алгоритмы прямого и обратного преобразований базируются на операциях с «рабочим отрезком».

Рабочим отрезком называется интервал [a; b] с расположенными на нем точками. Причем точки расположены таким образом, что длины образованных ими отрезков пропорциональны (или равны) частоте (вероятности) появления соответствующих символов.

Другой вариант алгоритма распаковки «сжатого» сообщения основан на свойстве рекуррентности прямого преобразования.

Производя на каждом шаге вычисление новых границ рабочего участка в соответствии с восстановленным на данном шаге символом (наподобие тех вычислений, которые мы выполняли при сжатии) и анализируя, на какой из отрезков этого участка попадает входное число (в нашем примере это 0,1071), восстанавливаем исходное сообщение.

**Листинг кода**

|  |
| --- |
| class Segment:  left = 0  right = 0  class SegmentDecode(Segment):  character = None  def define\_segments(letters, probability):  l = 0  segments = {letter: Segment() for letter in letters}  for i in range(len(letters)):  segments[letters[i]].left = l  segments[letters[i]].right = l + probability[i]  l = segments[letters[i]].right  return segments  def arithmetic\_encoding(letters, probability, s):  segments = define\_segments(letters, probability)  left = float(0)  right = float(1)  for i in range(len(s)):  symb = s[i]  new\_right = left + (right - left) \* segments[symb].right  new\_left = left + (right - left) \* segments[symb].left  left = new\_left  right = new\_right  return (left + right) / 2  def define\_segments\_decode(letters, probability):  l = 0  segments = [SegmentDecode() for i in range(len(letters))]  for i in range(len(letters)):  segments[i].left = l  segments[i].right = l + probability[i]  segments[i].character = letters[i]  l = segments[i].right  return segments  def arithmetic\_decoding(letters, probability, code, n):  segments = define\_segments\_decode(letters, probability)  s = ""  for i in range(n):  for j in range(len(letters)):  if segments[j].left <= code < segments[j].right:  s += segments[j].character  code = (code - segments[j].left) / (segments[j].right - segments[j].left)  break  return s  def main():  string = "летоисчисление"  probability = [string.count(i) / len(string) for i in (set(list(string)))]    encoded = arithmetic\_encoding(letters=list(set(list(string))), probability=probability, s=string)  print(encoded)  decoded = arithmetic\_decoding(letters=list(set(list(string))), probability=probability, code=encoded, n=len(string))  print(decoded)  main() |

**Вывод**: в данной работе был рассмотрен способ сжатия данных арифметическим методом. Данный метод позволяет представлять исходные данные в виде единственного числа меньше 1, полученного на основе вероятностей появления символов. Метод не является оптимальным, так как может выдавать несколько вариантов ответов, если вероятности некоторых символов повторяются.