Использование оптимальных кодов для сжатия данных: ZIPmeHuffman

Требования к программе

Требуется реализовать программу сжатия данных по методу Хаффмана

Программа должна работать в двух режимах.

Режим 1. На вход подаётся бинарный файл.

Программа сжимает файл, сохраняя результат в файл с тем же именем, что и входящий, но с расширением zmh.
Сжатие выполняется двухпроходным методом Хаффмена.
В первый проход строится модель, а во второй проход выполняется сжатие данных.
Формат сжатого файла определяется разработчиком программы.

Режим 2. На вход подаётся файл с расширением zmh

Программа выполняет разархивирование файла. Результат записывается в файл с тем же именем, что и входной файл, но без расширения zmh.
Предусмотреть возможно обработки файлов, которые повреждены или имеют некорректный формат. В этом случае надо уведомлять пользователя.

Предпочтительный язык программирования — python. В случае использования другого языка нужно создать простую сборочную систему на основе make-файлов для трёх операционных системы: Windows 10, MacOS 10.15+, Linux.

Пример работы программы. Пусть на вход программе был подан текстовый файл в UTF-8 кодировке:

```
мама мыла раму
```

Бинарное представление этого файла следующее:

```
d0bc("м") d0b0("а") d0bc("м") d0b0("а") 20(" ") d0bc("м") d18b("ы") d0bb("л")
d0b0("а") 20(" ") d180("р") d0b0("а") d0bc("м") d183("у")
```

Разбиваем файл на байты

```
d0 bc d0 b0 d0 bc d0 b0 20 d0 bc d1 8b d0 bb d0 b0 20 d1 80 d0 b0 d0 bc d1 83
```

Строим модель — словарь, содержащий сколько раз какой байт встречается в файле

```
{
  "d0": 9,
  "bc": 4,
  "b0": 4,
  "d1": 3,
  "20": 2,
  "8b": 1,
  "bb": 1,
  "80": 1,
  "83": 1
}
```

Строим код Хаффмена. Например такой,

```
{
  "d0": 0,
  "bc": 101,
  "b0": 110,
  "d1": 1111,
  "20": 1110,
  "8b": 10011,
  "bb": 10010,
  "80": 10001,
  "83": 10000
}
```

Получаем кодирование входного сообщений

```
0 101 0 110 0 101 0 110 1110 0 101 1111 10011 0 10010 0 110 1110 1111 10001 0 110 0 101 1111 10000
```

Или в байтах

```
01010110
01010110
11100101
11111001
10100100
11011101
11110001
01100101
11111000
0
```

Что тоже самое:

```
56 56 e5 f9 a4 dd f1 65 f8 00
```

Таким образом, на входе

```
d0bcd0b0d0bcd0b020d0bcd18bd0bbd0b020d180d0b0d0bcd183
```

получим сжатую версию файла:

```
5656e5f9a4ddf165f800
```

НО!

1. Нужно ещё добавить с этому построенный код Хаффмена, который представляет собой таблицу из девяти пар байтов! Соответственно нужно результат сжатия и таблицу «интегрировать» в один бинарный файл. Постарайтесь сделать это максимально компактно.
2. Последний байт является неполным! Надо это учесть. В противном случае при декодировании в файле появятся лишние байты и файл в итоге может быть испорчен.

На файлах очень маленького размера (десятки байт) накладные расходы на представление самого кода могут перекрывать выигрыш, полученный при сжатии содержимого. Это естественный процесс.

Требования к выполнению задания и принцип его оценивания.

1. Предполагается, что к программе будет приложен файл пояснительной записки с описанием формата zmh, объяснением как были решены проблемы с интеграцией кода в файлы и борьбы с возможными лишними битами в последнем байте. Также записка должна содержать анализ того на сколько можно добиться сжатия различных типов файлов. Необходимо провести аналитическое (ответить на вопрос почему наблюдаются такие результаты, как их можно интерпретировать) сравнение с известными архиваторами (zip, rar, 7z и т.д). Вклад в оценку — 25 %.
2. Оценивается правильность и скорость (!) работы программы (вклад в оценку — 70 %) и её эргономика (вклад в оценку — 5 %).
3. Плагиат кода или пояснительной записки — обнуляет выполнение задачи. И она не засчитывается. Плагиатом не является любое заимствование с указанием его авторства. Заимствованный участок кода не включается в оценивание. Например, Вы не смогли реализовать функцию и заимствовали её у друга, указав это в программе. Реализация этой функции исключается из вклада в оценку правильности работы программы на основе «важности» (принципиальности) этой функции в программе. Важность вещь разумно-субъективная, определяемая лектором курса. Так, если Вы заимствовали реализацию алгоритма сжатия, то, ясно, что это важная функция, т. к. она проверяет знания по теме курса. И её вклад может быть до 20 % вклада кода программы в оценку. В этом случае, этот вклад вычитается из общего вклада в 50 % и получается 30 % вклада программы. Если же была заимствована функция реализации интерфейса командной строки, то она не относится к тем функциям, которые призваны проверить знания по теме курса, поэтому её принципиальность может быть оценена не более, чем в 5 %.
4. Для получения оценки «отлично» нужно выполнить задание более, чем на 79 %, оценка «хорошо» ставится за выполнение на 65 % — 79 %, за 50 % — 65 % — «тройка», и при выполнении задания менее, чем на 50 % ставится — «неудовлетворительно».