Кодирование без потерь.

Займёмся кодированием без потерь. Каждую букву запишем с помощью нулей и единиц.
Разные символы могут быть записаны последовательностями разной длины.
Хотим, чтобы система кодирования позволяла однозначно расшифровать текст.

Если сразу считать буквы последовательностями чисел, то возникает задача сжатия последовательности.

Вопрос: Обсудите, можно ли написать программу, которая из любой последовательности нулей и единиц и делает более короткую последовательность, сохраняя возможность однозначной расшифровки. Обоснуйте Ваше мнение.

Как закодировать сообщения, чтобы частые сообщения передавать более экономно?
Как обеспечить, чтобы закодированное сообщение было легко распаковать?

Реализуемы два варианта:
* Кодирование с потерями, когда гарантируется сжатие всех последовательностей, но теряется возможность однозначно восстановить часть данных.
* Кодирование без потерь, когда сохраняется возможность восстановления, но не гарантируется, что закодированый текст будет короче исходного. Однако, мы можем стремиться, наиболее частые последовательности (буквы или фрагменты текста) сжимались, а редкие - увеличивались в размере.

Пример. Расмотрим алфавит A = { a, b, c, d } причем буквы встречаются с вероятностями P = {$^1/_2$, $^1/_4$, $^1/_8$, $^1/_8$}.
Код C_0. Закодируем каждый символ последовательностью из 0 и 1. Например,

буква	**код C_0**	**длина**
a	1000	4
b	0100	4
c	0010	4
d	0001	4

Чтобы закодировать последовательность символов, запишем подряд коды всех букв:
Код(abd) = 1000<u>0100</u>0001.

Кодирование без потерь означает, что разные исходные слова **всегда** отображаются в разные последовательности 0 и 1.

Например, код C_0 кодирует без потерь.

Как должен быть устроен код, чтобы расшифровка была легкой?

Удобство расшифровки

При расшифровке мы читаем закодированную последовательность 0 и 1 и решаем, из каких букв исходного алфавита она получилась.

Возьмём таблицу кодировки a - 10, b - 101, c - 1000, d - 10001 и будем раскодировать текст "10101...", получившееся из "abd". Какие возможны трудности?

Код буквы a совпадает с началом кода буквы b, это затрудняет расшифровку.
Прочтя начало текста "1010...", мы не уверены, где проходят границы букв: то ли исходное слово начинается на aa, то ли на ab, ac или ad. Чтобы это узнать, необходим прочесть закодированую последовательность дальше и отбросить невозможные варианты. Такой код не очень удобен для расшифровки.

Удобно, когда, прочтя код буквы, мы сразу можем ее распознать.

Пример неудобного кода: { 0, 10, 11, 100 } труден в распознавании, и это не случайно -- он кодирует с потерями.
Например, сообщение 010011 можно расшифровать несколькими способами:
"0 10 0 11" и "0 100 11".

Сформулируйте условие, чтобы это можно было сделать.

Такие коды называют префиксными -- от prefix, английского слова "приставка" .
http://ru.wikipedia.org/wiki/Префиксный_код

Префиксный код гарантирует кодирование без потерь: из разных текстов получатся разные коды.

Код { 0, 10, 11 } является префиксным. Поэтому сообщение 01001101110 можно разбить на слова единственным образом: 0 10 0 11 0 11 10.

Примеры.
Являются ли лёгкими для расшифровки следующие коды?
C_1 = { 0, 101 } - 0 не является началом 101. Код префиксный.
C_2 = { 1, 101 } - 1 является началом 101. Код не префиксный. Кодирует без потерь.
C_3 = { 0, 10, 110, 111 } - префиксный.
C_4 = { 00, 01, 10, 11 } - префиксный.

Код должен как можно сильнее сжать данные

При передаче сообщений, случайное событие - отправка текста.
Ожидаемая длина закодированного сообщения зависит от числа букв в тексте, от самого кода, и от частоты, с которой разные буквы встречаются в тексте.

Пусть наш алфавит $A = \{ a, b, c, d \}$,
буквы встречаются с вероятностями $P = \{ 1/2, 1/4, 1/8, 1/8 \} = \{ p_a, p_b, p_c, p_d \}$.
Использован код $C_3 = \{ 0, 10, 110, 111 \} = \{ C(a), C(b), C(c), C(d) \}$.

Буква	Вероятность P(буква)	код C_3	длина L_3
a	$^1/_2$	0	1
b	$^1/_4$	10	2
c	$^1/_8$	110	3
d	$^1/_8$	111	3

Длина каждого закодированного символа: L_3 = { 1, 2, 3, 3 } = { L(a), L(b), L(c), L(d) }. Найдите среднюю длину закодированного сообщения последовательностей 0 и 1, кодирующую сообщение из N = 4 букв.

L = (L(a) * $^1/_2$ + L(b) * $^1/_4$ + L(c) * $^1/_8$ + L(d) * $^1/_8$) * 4 =
= (1 * $^1/_2$ + 2 * $^1/_4$ + 3 * $^1/_8$ + 3 * $^1/_8$) * 4 = $^9/_4$ * 4 = 1.75 * 4 = 7.

Итак, на каждая буква потребует, в среднем, - 1.75 бита.
Сообщение из 4 букв - 7 бит.

Упражнения:
Тот же вопрос для кода C_4 = { 00, 01, 10, 11 }.
Тот же вопрос для кода C_5 = { 0, 1, 00, 11 }. Раскодируйте сообщение 000111000. Однозначна ли расшифровка?
Код C_6 = { 0, 10, 011, 111 } - получается из C_3 перестановкой символов. Сжатие, как у C_3: каждая буква - 1.75 бита. Сообщение из 4 букв - 7 бит. Однозначна ли расшифровка?

Максимальное сжатие, допускающее кодирование без потерь.
Пусть наш одну из букв кодирует символом "0", а остальные - тройками нудей и единиц. Сколько возможно таких троек в коде без потерь?
Для префиксного кода - тройка не может начинаться с "0". Остаются {100, 101, 110, 111}.

Если нам нужен код с большим количеством сиволов, нужно отказаться, например, от закодированного "100" (длины 3) и разрешить "1000" и "1001" (каждый длины 4).
При такой замене не меняется Сумма $2^{-(\text{длина закодированного символа})}$: у старого и у нового кода сумма одинакова:

$$\sum_{i \in C} 2^{-l_i}$$

Стартуем с кода из двух символов { 0, 1 }, который префиксный и однозначно раскодируем. Будем увеличивать количество и длину закодированных символов, как описано. Ненужные последовательности 0 и 1 выкинем. При этом код останется префиксным (и легко раскодируемым). Тогда длины символов всех слов удовлетворяют условию

$$\sum_{i \in C} 2^{-l_i} \leq 1.$$

Верно и обраное. Для любого набора длин, удовлетворяющего неравенству, можно построить подходящий префиксный код.

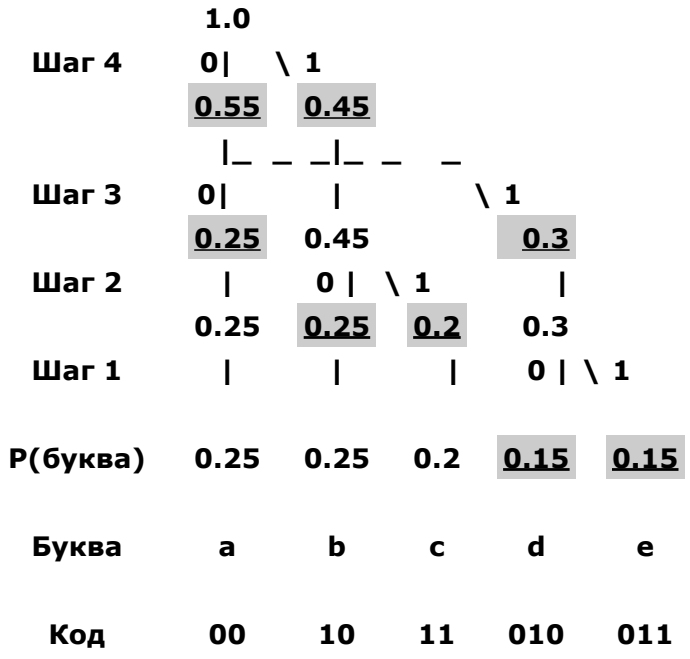
Код Хаффмана

http://algolist.manual.ru/compress/standard/huffman.php

Американский математик Давид Хаффман предложил способ построения префиксного кода.

1. Возмем два самых редко втречающихся символа в алфавите. Они будут закодированы самыми длинными последовательностями 0 и 1. Закодируем одно число 0, другое - 1.
2. Объединим эти два символа в один составной символ.
3. Продолжаем шаги 1 и 2, каждый раз наращивая код символа спереди.

Пример.
Пусть наш алфавит $A = \{ a, b, c, d, e \}$.
и буквы встречаются с вероятностями $P = \{ {}^{1}/_{4}, {}^{1}/_{4}, {}^{1}/_{5}, {}^{3}/_{20}, {}^{3}/_{20} \}$.
Тогда получится код $C = \{ 00, 10, 11, 010, 011 \}$.



Сосчитайте среднюю длину закодированного текста из *N* букв; среднюю длину кода, приходящегося на 1 букву текста.

Не очень простая задача. Предположите, что есть более экономный префиксный код. Постарайтесь получить противоречие или ограничить возможные примеры таких кодов. Например, можно ли сэкономить, по-другому установив соответствие между буквами и найденными кодами букв? Пусть у него два символа с наименьшей вероятностью имеют другие длины, чем в коде Хаффмана. Покажите, что тогда код будет иметь большую

среднюю длину.

Код Хаффмана несколько страннен тем, что построение ведётся снизу вверх.
Если вести построение сверху вниз, разделяя алфавит на группы с примерно равной вероятностью, можно получить префиксный код. Но средняя степень сжатия закодированного символа не будет меньше.

Количество информации.

Дополнение -- не вошло в лекцию.

Вспомним развлечение для первоклассников: за сколько вопросов можно отгадать задуманное число, скажем, от нуля до 15? На каждый вопрос нам отвечают "да" или "нет".
Вот последовательность вопросов, понятная ученикам для 9 класса:
1: $x \geq 8$?
2: $x \bmod 8 \geq 4$?
3: $x \bmod 4 \geq 2$?
4: $x \bmod 2 = 1$?

Упражнение. Если записать подряд ответы на эти вопросы (да - 1, нет - 0), получится двоичная запись числа x. Проверьте для числа 13.

Любая последовательность из четырёх 0 и 1 задает загаданное число, $2^4 = 16$ вариантов.
Если числа загадывают случайно с одинаковой вероятностью $p=1/2^4$, то нужно задать все 4 вопроса, и нельзя придумать более эффективного алгоритма для отгадывания.

Если у загадывающего есть 2 любимых числа, например, 1 и 5, то было бы достаточно одного вопроса, чтобы узнать, что задумано.

В общем случае числа могут быть загаданы с разными вероятностями: $p_1=p_5=0.35$, а остальные двенадцать -- $p_i = 0.3/12 = 3/120 = 1/40 = 0.025$.

Как в этом случае отгадать ответ наиболее эффективно?
Если мы собираемся отгадывать много раз, и право задать каждый вопрос что-то стоит, мы будем стремиться уменьшить среднюю стоимость отгдадывания.

Мы по-прежнему можем гарантированно отгадать задуманное число за 4 вопроса.
Как использовать наше знание о вероятностях, чтобы в среднем отгадать число быстрее?

Количеством информации связано со средним ожидаемым количеством вопросов с ответами "да" или "нет", которые необходимы для определения ответа -- при наиболее эффективном способе спрашивать, с учётом вероятностей разных исходов.

Точное определение я сейчас избегаю давать. Сейчас можем считать, что количество инфолрмации - это среднее ожидаемое количество вопросов.

Сравним два текста: в одном все буквы встречаются с одинаковой вероятностью, а второй текст - фраза на русском языке.

Если мы удалим букву из случайного текста, мы потеряем часть информации и не сможем восстановить её.
Если мы удалим часть русского текста, то, скорее всего, сможем догадаться, что было удалено. Поэтому каждая отдельная буква русского текста содержит меньше информации, текст более предсказуем.

Согласуется ли это с определением количекства информации как количества вопросов, необходимых для отгадывания: для чего нужно больше вопросов: для отгадывания случайноготекста или русского текста?

Количество информации и кодирование с потерями

Займёмся отгадыванием текстов, в которых разные буквы встречаются с разными вероятностями.

Давайте согласимся на неполное восстановление текста. Например, пожертвуем наиболее редкими буквами.

Пример: возьмём часть английского алфавита, первые $2^3 = 8$ букв, $A = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ \mathrm{d}, \ \mathrm{e}, \ \mathrm{f}, \ \mathrm{g}, \ \mathrm{h}\}$. Пусть вероятности встретить их в тексте равны $P = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{3}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}\right\}$.

Если забыть про вероятности (считать их равными), для определения одной буквы нужно три вопроса. Количество информации в каждой букве равно 3 битам. Каждую букву можно задать тройкой нулей и единиц, например: a - 000, b - 001, c - 010, ... g - 110, h - 111.

Довольно высоки шансы, что случайная буква находится в первой половине алфавита: $P(x \in \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}) = 15/16$. Если согласиться на риск, что в d = 1/16 случаев мы не сможем восстановить букву, то можно будет обойтись 4 буквами и более коротким кодом, использующим 2 бита на букву: a - 00, b - 01, c - 10, d - 11.

Количество информации при допустимой доле ошибок d - среднее ожидаемое количество вопросов, которые необходимо задать для восстановления текста при допустимой вероятности ошибки d.

Попробуем восстанавливать тексты.
Возьмём алфавит из двух букв: 0 и 1. Их вероятности равны $p_0 = 0.9$ и $p_1 = 0.1$.
Текст размера $N = 100$.

* Сосчитайте вероятность строки x = 1011010010...101, в которой k единиц и $N - k$ нулей.
ОТВЕТ
$P(x) = p_1^k \ p_0^{N-k}$.
Найдите вероятность, что в строке все 0. Что все 1.

* Сколько существует разных строк x длины N, в каждой из которой k единиц и $N - k$ нулей?

Таким образом, вероятность встретить **произвольную** строку из k единиц и $N - k$ нулей равна $C_N^k \ p_1^k \ p_0^{N-k}$.
Как эта вероятность зависит от числа еднинц k?

Пояснение

* Вероятность встретить **произвольную** строку из k единиц и $N - k$ нулей $C_N^k \, p_1^k \, p_0^{N-k}$ максимальна при $k = N \, p_1$.

* Ширина максимума примерно равна $\sqrt{N \, p_1 (1 - p_1)}$. Это значит, что диапазон k, при которых вероятность близка к максимальной примерно равен $N \, p_1 \pm \sqrt{N \, p_1 (1 - p_1)}$.

Пример: N = 1000, p_1 = 0.1, 1-p = 0.9.
Ожидаемый диапазон $k \sim 1000 \cdot 0.1 \pm \sqrt{1000 \cdot 0.1 \cdot 0.9} \approx 100 \pm 10$ единиц.
Шансы встретить последовательность, включающую 150 или более единиц на 1000 малы.

При кодировании с потерями достаточно позаботиться только о часто встречающихся последовательностях. Поэтму можно сделать код, который позволит восстанавливать только последовательности, включающие 150 или менее единиц на каждую тысячу.

С другой стороны, если наш код будет не в состоянии описывать типичные последовательности, включающие примерно 100 единиц на 1000 символов, то мы почти никакой текст не сможем закодировать.