Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Поволжский государственный университет телекоммуникаций и информатики»

МЕТОДИЧЕСКАЯ РАЗРАБОТКА

к лабораторным работам
по дисциплине
«Теория информации»

Направление (специальность) подготовки	09.03.01 Информатика и вычислительная техника
Профиль (специализация) подготовки	Программное обеспечение средств вычислительной техники и автоматизированных систем
Направление (специальность) подготовки	09.03.01 Информатика и вычислительная техника (индивидуальная форма обучения)
Профиль (специализация) подготовки	Программное обеспечение средств вычислительной техники и автоматизированных систем
Направление (специальность) подготовки	09.03.04 Программная инженерия
Профиль (специализация) подготовки	Разработка программно-информационных систем
Направление (специальность) подготовки	02.03.03 Математическое обеспечение и администрирование информационных систем
Профиль (специализация) подготовки	Технология программирования

Самара
2016

Лабораторная работа №1 «Исследование информационных характеристик источников и алгоритмов сжатия данных»

Порядок выполнения лабораторной работы

Начальная подготовка

Включить компьютер. Выбрать пользователя «Студент». После этого, компьютер будет перейдёт к программной оболочке «Volkov Commander» (или файловый менеджер «Far») с вызовом пользовательского меню «user menu». Отказаться от вызова меню, нажав клавишу «ESC». Для входа в меню «Volkov Commander» (файловый менеджер «Far») нажмите «F9» и стрелку вниз «↓» (или «Enter»). Появится выпадающее меню. Переход к соседним выпадающим меню с помощью клавиш «→» и «←». Переход к пунктам выпадающего меню с помощью клавиш «↓» и «↑».

Выбор пункта выпадающего меню клавиша «Enter». Выход из меню — «Esc».

Установите полный режим просмотра в обоих окнах:

Нажмите «F9» переход к «Left» «↓» (появится выпадающее меню). В выпадающем меню выбрать пункт «Full» и нажать «Enter».

Нажмите «F9» переход к «Right» «↓». В выпадающем меню выбрать пункт «Full» и нажать «Enter».

Нажмите «F9» переход к «Right» «↓». В выпадающем меню выбрать пункт «Drive…» (горячая клавиша «Alt+F2»). В появившемся меню «Drive letter» выберете диск с буквой «D» и нажмите «Enter».

С помощью стрелок «↓» и «↑» найдите каталог «USERS», и установите на нём курсор. Нажав «Enter» войдите в него. Дальше надо создать Ваш рабочий каталог. Для этого нажмите «F9» переход к «Files» «↓».. В выпадающем меню выбрать пункт «Make Directory» (горячая клавиша «F7», также есть краткая подсказка внизу экрана перед входом в меню файлового менеджера). В появившемся меню «Make Directory»→«Create the directory» набрать имя Вашего каталога. Имя должно быть не более 8 латинских символов или цифр. Допускаются знаки «–» и «_». Пробел « » не допускается! Набрав имя каталога нажмите «Enter». Установить курсор на имя Вашего каталога и нажмите «Enter». В этом случае Вы окажетесь в своём каталоге, где будут размещаться Ваши файлы.

Задание 1.1

Для начала работы Вам необходимо создать новый файл. Нажмите «Shift+F4». Появится меню «Edit»→«Edit the file». В появившемся меню набрать имя файла «1.txt» и нажать «Enter». Если появится сообщение «Can't find the file» (Не могу найти файл), то в ответ на это нажмите клавишу «Enter».

В окне появится простейший редактор, в котором Вам необходимо набрать в строчку 100 символов. При этом клавишу «Enter» не нажимать. Среди набираемых 100 символов должно быть 50 символов «0», 35 символов «1» и 15 символов «2». (В верхнем правом углу можно проконтролировать количество набранных символов после слова «Col»). Сохранить файл, нажав клавишу «F2» и выйти из редактора, нажав клавишу «Esc» (или «F10»). В вашем каталоге должен появиться файл с именем «1», расширением «txt» и размеров в 100 байт. Просмотр файла можно сделать следующим образом.

Для этого нажмите «F9» переход к «Files» «↓».. В выпадающем меню выбрать пункт «View» (горячая клавиша «F3», также есть краткая подсказка внизу экрана перед входом в меню «Volkov Commander»). Для перехода в просмотр 16-ричных кодов используется клавиша «F4» (возврат в текстовый режим просмотра тоже клавиша «F4»). При просмотре слева по 8 символов идут адреса байт в файле. Далее непосредственно строчка из 16 байт (коды ASCII). Справа их символьное отображение. Убедитесь, что в Вашем файле имеются только коды «30», «31» и «32», которые соответствуют символам «0», «1» и «2». Если там есть посторонние коды, то надо выйти из просмотра «Esc» (или «F10»), установить курсор на файл «1.txt» и нажать «F4». Тем самым Вы перейдёте в редактор и сможете отредактировать файл. Удалять символы можно с помощью клавиш «Backspace» (удаление символа слева от курсора) или «Del» (удаление символа, на который указывает курсор). Поправив текст, сохраните его: «F2». И выйдите из редактора: «Esc». Опять просмотром проверьте правильность текста. Если в результате исправлений опять останутся иные символы, кроме вышеупомянутых, то позовите преподавателя.

Если файл сделан благополучно, то внизу в командной строке наберите команду:

```
usl_ver.exe 1.txt
```

и нажмите клавишу «Enter».

Для одновременной работы и просмотра выполните следующее.

Нажмите «F9» переход к «Left» «↓».. В выпадающем меню выбрать пункт «On/Off» (горячая клавиша «Ctrl+F1») и нажмите «Enter». У Вас левая панель исчезнет, а правая останется. На месте левой панели Вы сможете увидеть результат выполнения того, что Вы набрали в командной строке. Если правая панель всё же мешает полному прочтению, то Вы временно можете убрать и её. Нажмите «F9» переход

к «Right» «↓».. В выпадающем меню выбрать пункт «On/Off» (горячая клавиша «Ctrl+F2») и нажмите «Enter». Восстановление правой панели потребует тех же действий: «F9» переход к «Right» «↓» — «On/Off» «Enter».

Правильность файла подтвердиться сразу после набранной команды первой строчкой:

```
[0]='0' [1]='1' [2]='2'
```

С тёмного экрана записать результат выполнения программы, которая протестировала Ваш набранный файл «1.txt».

1. Условные вероятности.
2. Безусловные вероятности.
3. Энтропию и избыточность для найденных безусловных частостей.
4. Возможность сжатия синтезированного текста без учёта связи между соседними символами.
5. Энтропию и избыточность для найденных условных частостей.
6. Максимальную энтропию.
7. Возможность сжатия синтезированного текста при учёте связи между соседними символами.

Создайте файл «m.txt» и запишите туда значения условных вероятностей, по три в строчке, разделяя их пробелом (или символом табуляции). При этом численные значения округлите до 2 знаков, после десятичной точки. Сохраните файл и выполните следующую команду:

```
gen_ver.exe m.txt 2.txt 48 100000
```

и нажмите клавишу «Enter».

Запишите

1. Исходную таблицу условных вероятностей.
2. Рассчитанные вероятности.
3. Энтропию и избыточность для рассчитанных безусловных вероятностей.
4. Максимальную энтропию
5. Оценку возможности сжатия исходного текста без учёта связей между символами.
6. Энтропия и избыточность для рассчитанных условных вероятностей.
7. Оценку возможности сжатия исходного текста при учёте связи между соседними символами.
8. Количество выпадений символов согласно таблице условных вероятностей.
9. Таблицу условных частостей.
10. Частости выпадений.
11. Энтропию и избыточность для найденных безусловных частостей.
12. Возможность сжатия синтезированного текста без учёта связи между соседними символами.
13. Энтропию и избыточность для найденных условных частостей.
14. Возможность сжатия синтезированного текста при учёте связи между соседними символами.

Если среди выпавших частостей встречаются нули, то повторите эксперимент, набрав команду

```
gen_ver.exe m.txt 2.txt 48 100000
```

Повторяйте эксперимент, если среди частостей есть хотя бы одна нулевая. Добейтесь того, что бы все частости были ненулевые.

При успешном эксперименте запишите:

1. Количество выпадений символов согласно таблице условных вероятностей:
2. Таблицу условных частостей:
3. Частости выпадений:
4. Энтропию и избыточность для найденных безусловных частостей:
5. Возможность сжатия синтезированного текста без учёта связи между соседними символами:
6. Энтропию и избыточность для найденных условных частостей:
7. Возможность сжатия синтезированного текста при учёте связи между соседними символами.

Найдите у себя в каталоге файл «2.txt» и просмотрите его на особенность появления символов в файле.

Сделайте вывод.

Задание 1.2

Синтезируем матрицу условных вероятностей 4x4 для алфавита, состоящего из четырёх символов: «0», «1», «2» и «3». Синтезируем текст из этих символов, с учётом полученной матрицы условных вероятностей. Определим информационные характеристики для синтезированного файла. Для этого в командной строке наберите следующие команды (нажимая «Enter» после каждой команды)

```
gen_matr.exe 4 > m2.txt
gen_ver.exe m2.txt 3.txt 48 100000
```

Запишите

1. Исходную таблицу условных вероятностей.
2. Рассчитанные вероятности.
3. Энтропию и избыточность для рассчитанных безусловных вероятностей.
4. Максимальную энтропию
5. Оценку возможности сжатия исходного текста без учёта связей между символами.
6. Энтропию и избыточность для рассчитанных условных вероятностей.
7. Оценку возможности сжатия исходного текста при учёте связи между соседними символами.
8. Количество выпадений символов согласно таблице условных вероятностей.
9. Таблицу условных частостей.
10. Частости выпадений.
11. Энтропию и избыточность для найденных безусловных частостей.
12. Возможность сжатия синтезированного текста без учёта связи между соседними символами.
13. Энтропию и избыточность для найденных условных частостей.
14. Возможность сжатия синтезированного текста при учёте связи между соседними символами.

Сравните исходную таблицу условных вероятностей и таблицу условных частостей. Сделайте вывод по условиям сравнения.

Рассчитайте объём текста в битах при равномерном кодировании символов по формуле $V = S \cdot L$, где S — размер исходного текста в байтах;
L — среднее количество бит на 1 символ при условии, что все символы равновероятны и независимы.

Определить, какой длины в байтах получится сжатый текст с учётом связей между соседними символами.

В командной строке наберите команду

```
gen_ver.exe m2.txt 4.txt 48 100
```

Запишите
1. Количество выпадений символов согласно таблице условных вероятностей.
2. Таблицу условных частостей.
3. Частости выпадений.

Сравните исходную таблицу условных вероятностей, записанную Вами в предыдущем случае и таблицу условных частостей, полученной в данном эксперименте. Сделайте вывод по условиям сравнения. Поясните различия сравнения исходной таблицы условных вероятностей с таблицей условных частостей для первого эксперимента (файл 3.txt — 100000 байт) и второго — (файл 4.txt — 100 байт).

Задание 1.3

Оценить степень сжатия современного архиватора. Для этого в командной строке наберите следующую команду

```
7z.exe a 3.7z 3.txt
```

В итоге в Вашем каталоге появится архивный файл **`3.7z.`** Необходимо определить размер сжатых данных в этом файле. Для этого установите курсор на файл **`3.7z`** и просмотрите файл в 16-ричных кодах («F3» «F4»). Запишите последние 4 байта верхней строки (по адресам 000000C, 000000D , 000000E, 000000F) в обратном порядке. В результате должно получиться 8 разрядное 16-ричное число. Переведите его в десятичную систему счисления. Полученное значение — размер шифрованных данных в архиве **`3.7z`** в байтах. Переведите это значение в биты, умножив это количество байт на 8.

Оцените степень сжатия по сравнению с возможным сжатием текста, сформированного в файле **`3.txt`** при учёте связи между соседними символами, найдя отношение между полученной величиной сжатия от архиватора 7z и теоретической величиной длины в битах сжатого текста в файле **`3.txt`** с учётом связей между соседними символами.

Задание 1.4. Алгоритм Хаффмана

Для начала работы Вам необходимо создать новый файл. Нажмите «Shift+F4». Появится меню «Edit»→«Edit the file». В появившемся меню набрать имя файла «2_1.txt» и нажать «Enter». Если появится сообщение «Can't find the file» (Не могу найти файл), то в ответ на это нажмите клавишу «Enter». В окне появится простейший редактор, в котором Вам необходимо набрать в строчку символы от «0» до «7», соответствующие вероятностям, взятым из нижеследующей таблицы.

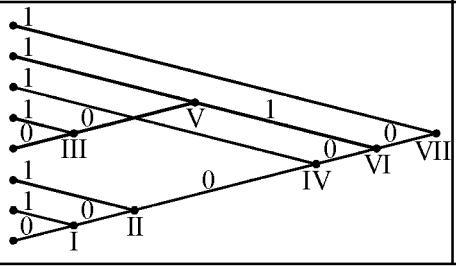
Символ источника	Вероятность $p(a_k)$	Кодовое дерево	Код	n_k	$n_k p(a_k)$
a_1	0,40		1	1	0,40
a_2	0,13		011	3	0,39
a_3	0,12		001	3	0,36
a_4	0,11		0101	4	0,44
a_5	0,11		0100	4	0,44
a_6	0,08		0001	4	0,32
a_7	0,03		00001	5	0,15
a_8	0,02		00000	5	0,10

Рис. 1.1

Сохранить файл, нажав клавишу «F2» и выйти из редактора, нажав клавишу «Esc» (или «F10»). В вашем каталоге должен появиться файл с именем «2_1», расширением «txt».

Для сжатия исходного файла «2_1.txt» выполните следующую команду.

```
kod_f.exe 2_1.txt 2_1.arh
```

Запишите кодированную часть в битах для данного источника информации. Определите среднюю длину в битах для результирующей кодовой последовательности, разделив количество бит в кодированном сообщении на количество символов в исходном файле **`2_1.txt`**.

Сравните полученную таблицу кодов с таблицей, приведённой на рисунке.

Для записи безусловных вероятностей выполните команду

```
usl_ver.exe 2_1.txt
```

В полученном выводе найдите сообщение о выводе безусловных вероятностей и запишите их. Найдите энтропию и среднюю длину экономного кода. Сравните расчётную и экспериментальную средние длины экономного кода.

Для раскодировки файла выполните следующую команду

```
dek.exe 2_1.arh 2_1_.txt
```

Сравните исходный и раскодированный файлы. Для сравнения выполните следующую команду

```
comp.exe 2_1.txt 2_1_.txt
```

Сгенерируйте новый файл **`3.txt`** с помощью последовательности следующих команд (или возьмите уже готовый из лабораторной работы №1)

```
gen_matr.exe 4 > m2.txt
gen_ver.exe m2.txt 3.txt 48 100000
```

Сожмите файл, выполнив следующую команду

```
kod_h.exe 3.txt 3_h.arh
```

Запишите кодированную часть в битах для результирующего файла **`3_h.arh`**. Определите среднюю длину в битах для результирующей кодовой последовательности, разделив количество бит в кодированном сообщении на количество символов в исходном файле **`3.txt`**.

Запишите таблицу кодов. Для записи безусловных вероятностей выполните команду

```
usl_ver.exe 3.txt
```

В полученном выводе найдите сообщение о выводе безусловных частостей, энтропию для найденных безусловных частостей, возможность сжатия синтезированного текста без учёта связи между соседними символами, энтропию для найденных условных частостей, возможность сжатия синтезированного текста с учётом связи между соседними символами и запишите их. Найдите энтропию и среднюю длину экономного кода. Сравните расчётную и экспериментальную средние длины экономного кода. Определите во сколько раз полученная средняя длина полученного экономного кода больше теоретической.

Задание 1.5. Алгоритм Шеннона-Фано

Для исследования сжатия по данному алгоритму воспользуемся файлом **`2_1.txt`**, созданным в задании 1. Таблица безусловных вероятностей, такая же как и в задании 1, приведена ниже на рисунке

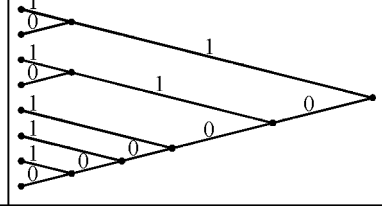
Символ источника	Вероятность $p(a_k)$	Кодовое дерево	Код	n_k	$n_k p(a_k)$
a_1	0,40		11	2	0,80
a_2	0,13		10	2	0,26
a_3	0,12		011	3	0,36
a_4	0,11		010	3	0,33
a_5	0,11		001	3	0,33
a_6	0,08		0001	4	0,32
a_7	0,03		00001	5	0,15
a_8	0,02		00001	5	0,10

Рис. 1.2

Для сжатия исходного файла «2_1.txt» выполните следующую команду.

kod_sf.exe 2_1.txt 2_2.arh

Запишите кодированную часть в битах для данного источника информации. Определите среднюю длину в битах для результирующей кодовой последовательности, разделив количество бит в кодированном сообщении на количество символов в исходном файле **2_1.txt**.

Сравните полученную таблицу кодов с таблицей, приведённой на рисунке.

Для раскодировки файла выполните следующую команду

dek.exe 2_2.arh 2_2_.txt

Сравните исходный и раскодированный файлы. Для сравнения выполните следующую команду

comp 2_2.txt 2_2_.txt

Сожмите файл **3.txt**, созданный в 1 задании, выполнив следующую команду

kod_sf.exe 3.txt 2_sf.arh

Запишите кодированную часть в битах для результирующего файла 2**_sf.arh**. Определите среднюю длину в битах для результирующей кодовой последовательности, разделив количество бит в кодированном сообщении на количество символов в исходном файле **3.txt**.

Используйте записанную в 1 задании таблицу безусловных вероятностей для файла **3.txt** для нахождения энтропии и средней длины экономного кода. Сравните расчётную и экспериментальную средние длины экономного кода. Определите во сколько раз средняя длина полученного экономного кода больше теоретической, определённой в задании 1.

Задание 1.6. Алгоритм LZ77

Аналогично действиям из задания 1 создайте файл «3_1.txt». В этом файле заглавными буквами наберите фразу «КРАСНАЯ КРАСКА». Для сжатия набранного сообщения выполните команду

LZ77LZSS.exe 3_1.txt 2_3.lz7 8 5

Размер кодированного сообщения в битах определить по формуле $R = \lfloor (S-4)\cdot 8/14 \rfloor \cdot 14$, где S — полученный размер архивного файла 2**_3.lz7**. Сравнить этот размер с размером, полученным из таблицы

СЛОВАРЬ(8)	БУФЕР(5)	КОД
«........»	«КРАСН»	<0,0‘К’>
«.......К»	«РАСНА»	<0,0‘Р’>
«......КР»	«АСНАЯ»	<0,0‘А’>
«.....КРА»	«СНАЯ »	<0,0‘С’>
«....КРАС»	«НАЯ К»	<0,0‘Н’>
«...КРАСН»	«АЯ КР»	<5,1‘Я’>
«.КРАСНАЯ»	« КРАС»	<0,0‘ ’>
«КРАСНАЯ »	«КРАСК»	<0,4‘К’>
«АЯ КРАСК»	«А....»	<0,0‘А’>

Рис. 1.3

Сжать файл **3.txt**, выполнив следующую команду

LZ77LZSS.exe 3.txt 3_3.lz7 8 5

Определить по вышеприведённой формуле размер кодированного сообщения в битах. Сравните расчётную и экспериментальную средние длины экономного кода. Определите во сколько раз средняя длина полученного экономного кода больше теоретической, определённой в задании 1. Запишите размер архивного файла **3_3.lz7**.

Получите исходный файл, выполнив команду

LZ77LZSS.exe 3_3.lz7 3_3_.txt

Сравните исходный и результирующий файлы командой

comp 3.txt 3_3_.txt

Сделайте вывод о результатах сравнения.

Сожмите файл при других значениях размера словаря при ограниченном и неограниченном буфере. Определите, как влияет размер буфера на размер получаемого архива. Определите, как влияет размер словаря при неограниченном буфере на размер получаемого архива.

Задание 1.7. Алгоритм LZSS

Для сжатия по заданному алгоритму сообщения «КРАСНАЯ КРАСКА», созданного в предыдущем задании, выполните команду

LZ77LZSS.exe 3_1.txt 2_4.lzs 8 5

Приблизительный размер кодированного сообщения в битах определить по формуле $R = (S-4)\cdot 8$, где S — полученный размер архивного файла 2**_4.lzs**. Сравнить этот размер с размером, полученным из таблицы

СЛОВАРЬ(8)	БУФЕР(5)	КОД	ДЛИНА КОДА
«........»	«КРАСН»	0‘К’	9
«.......К»	«РАСНА»	0‘Р’	9
«......КР»	«АСНАЯ»	0‘А’	9
«.....КРА»	«СНАЯ »	0‘С’	9
«....КРАС»	«НАЯ К»	0‘Н’	9
«...КРАСН»	«АЯ КР»	1<5,1>	7
«..КРАСНА»	«Я КРА»	0‘Я’	9
«.КРАСНАЯ»	« КРАС»	0‘ ’	9
«КРАСНАЯ »	«КРАСК»	1<0,4>	7
«НАЯ КРАС»	«КА...»	1<4,1>	7
«АЯ КРАСК»	«А....»	1<0,1>	7

Рис. 1.4

Сжать файл **3.txt**, выполнив следующую команду

LZ77LZSS.exe 3.txt 3_4.lzs 8 5

Определить по вышеприведённой формуле размер кодированного сообщения в битах. Сравните расчётную и экспериментальную средние длины экономного кода. Определите во сколько раз средняя длина полученного экономного кода больше теоретической, определённой в задании 1. Запишите размер архивного файла **3_4.lzs**.

Получите исходный файл, выполнив команду

LZ77LZSS.exe 3_4.lzs 3_4_.txt

Сравните исходный и результирующий файлы командой

comp 3.txt 3_4_.txt

Сделайте вывод о результатах сравнения.

Сожмите файл при других значениях размера словаря при ограниченном и неограниченном буфере. Определите, как влияет размер буфера на размер получаемого архива. Определите, как влияет размер словаря при неограниченном буфере на размер получаемого архива.

Задание 1.8. Алгоритм LZ78

Для сжатия по заданному алгоритму сообщения «КРАСНАЯ КРАСКА», созданного в задании 3, выполните команду

LZ78.exe 3_1.txt 2_5.lz8 0

Записать размер кодированного сообщения («число кодированных бит»). Сравнить этот размер с размером, полученным из таблицы

ВХОДНАЯ ФРАЗА (В СЛОВАРЬ)	КОД	ПОЗИЦИЯ СЛОВАРЯ
«»		0
«К»	<0‘К’>	1
«Р»	<0‘Р’>	2
«А»	<0‘А’>	3
«С»	<0‘С’>	4
«Н»	<0‘Н’>	5
«АЯ»	<3‘Я’>	6
« »	<0‘ ’>	7
«КР»	<1‘Р’>	8
«АС»	<3‘С’>	9
«КА»	<1‘А’>	10

Рис. 1.5

Сжать файл **3.txt**, выполнив следующую команду

LZ78.exe 3.txt 3_5.lz8 0

Записать размер кодированного сообщения («число кодированных бит»). Сравните расчётную и экспериментальную средние длины экономного кода. Определите во сколько раз средняя длина полученного экономного кода больше теоретической для кодирования с учётом связи между соседними символами. Запишите размер архивного файла **3_5.lz8**. Запишите размер полученного словаря.

Получите исходный файл, выполнив команду

```
LZ78.exe 3_5.lz8 3_5_.txt
```

Сравните исходный и результирующий файлы командой

```
comp 3.txt 3_5_.txt
```

Сделайте вывод о результатах сравнения.

Для каждого из трёх вариантов очистки словаря сожмите файл при значениях размера словаря, начиная от 1 до размера, превышающего полученный в предыдущем примере, увеличивая размер словаря каждый раз в 2 раза. Записывайте полученные размеры кодированной части архивного файла.

Примеры командных строк:

```
LZ78.exe 3.txt 3_6.lz8 512 0
LZ78.exe 3.txt 3_7.lz8 512 1
LZ78.exe 3.txt 3_8.lz8 512 2
```

Здесь предпоследней цифрой задаётся размер словаря (в данном случае 512 фраз), а последней — вариант очистки словаря:

0 — при переполнении словаря очищать словарь,

1 — при переполнении словаря оставлять односимвольные последовательности,

2 — при переполнении словаря удаление из словаря кода с наименьшим количеством ссылок.

Постройте на одном графике зависимости изменения длины архива для каждого из трёх вариантов.

Сделайте вывод по полученным зависимостям. Определите наименьший размер кодированной части архива. Сравните, во сколько раз теоретическое значение длины кода будет отличаться от полученного наименьшего размера кодированной части.

Сравните, полученный минимальный размер кодированной части с аналогичным размером архиватора 7z.

Задания для самостоятельного моделирования по теме «Исследование информационных характеристик источников и алгоритмов сжатия данных, а также помехоустойчивого кодирования»

Лабораторная работа №2. Генератор символов

Целью выполнения приведённых ниже заданий заключается в приобретении навыков студентами очного отделения путём компьютерного моделирования отдельных разделов теории информации. Компьютерное моделирование заключается в создании студентами программного кода. Выбор языка программирования, как инструмента для моделирования, предлагается сделать студентами самостоятельно. При моделировании студент должен показать способность самостоятельной разработки программного алгоритма, а также его реализации в виде программного кода. При этом, использование чужих библиотек, реализующих предлагаемые алгоритмы не допускается. Результатом выполнения заданий является написанный программный код, пояснение этого кода, демонстрация правильности работы программы, алгоритм, исходные и результирующие данные и необходимые графические построения в виде отчёта по каждому заданию.

На рис. 2.1 приведена блок-схема, в которой поясняются этапы и последовательность выполнения заданий. 2.1 и 2.2.

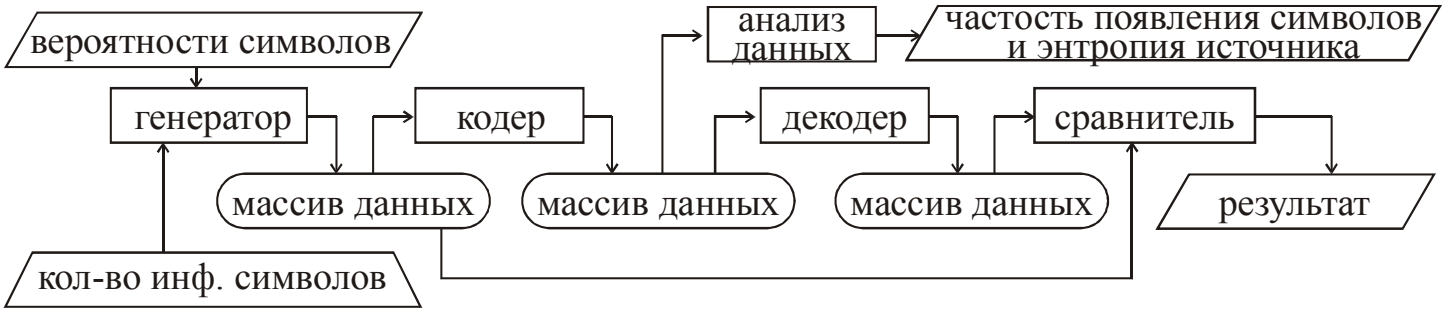


Рис. 2.1. Блочная модель для проверки кодера и декодера (для шифрования и экономного кодирования)

Задание 2.1.

Реализовать программным способом генератор независимых символов.

На рис. 1. прямоугольником обозначены исполняемые модули. Исполняемый модуль «генератор» имеет параметры. Блоком в виде параллелограмма, стрелка от которого идёт на блок «генератор», обозначен список вероятностей символов. Этот список необходимо иметь в виде текстового файла. Если при программировании вероятности задаются в оконном интерфейсе, то предусмотреть возможность сохранения введённых параметров в текстовом файле. Трапецией, стрелка от которого также идёт на блок «генератор», обозначен численный параметр, который характеризует число генерируемых символов (объём выборки). Рекомендуемый минимальный размер выборки составляет 10^6. Блок с закруглёнными сторонами, к которому идёт стрелка от блока «генератор» представляет собой массив данных, содержащий генерируемые символы.

Формат данных и рекомендуемое расширение файлов, содержащих результирующий массив данных, приведён на рис. 2.2 в виде скриншотов части экрана монитора при просмотре файлов в 16-ричнном формате в файловом менеджере «Far». Левая колонка представляет собой численные значения адресов в 16-ричнной системе счисления, оканчивающиеся двоеточием. В центральной части — 16-ричное представление байтов в файле, по 16 в строке. Для удобства восприятия, отображение байтов поделено в середине пополам сплошной линией. Правая колонка отображает символьное отображение каждого байта также по 16 в строке (без разделительных пробелов).

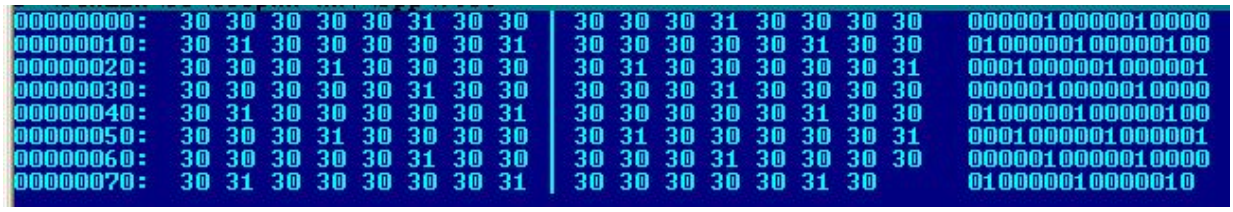
Расширение «cb» (от слов «char binary») — представляет собой обозначение бинарной формы представления данных, численно соответствующих беззнаковым 8-разрядным целым числам, имеющим диапазон значений 0…255.

Расширение «ct» (от слов «char text») — представляет собой обозначение текстовой формы представления данных в виде байтовых символов (октетов) из таблицы ASCII.

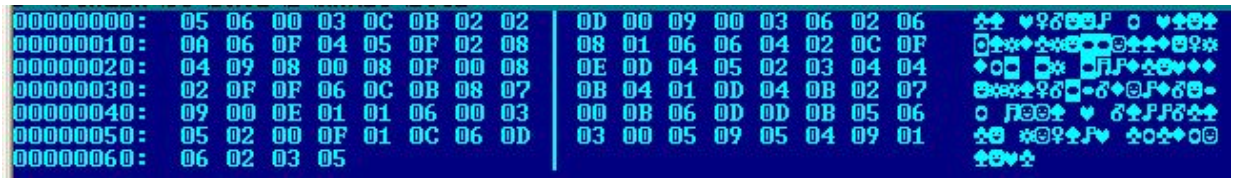
Генератор должен генерировать произвольное количество вариантов уникальных символов m. Поэтому требуется определить соответствие вероятности каждому уникальному генерируемому символу. Способы этого соответствия выбираются студентом самостоятельно.

Бинарное представление двоичной информации: формат «cb»

Бинарное представление двоичной информации: формат «ct»



Бинарное представление шестнадцатеричной информации: формат «cb»



Бинарное представление шестнадцатеричной информации: формат «ct»

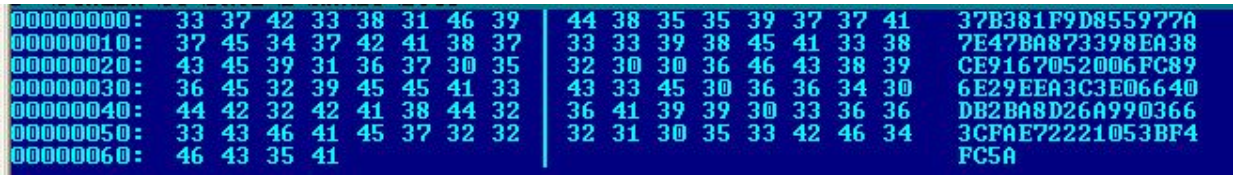


Рис. 2.2. Бинарное представление предлагаемых форматов данных

Примеры соответствия:

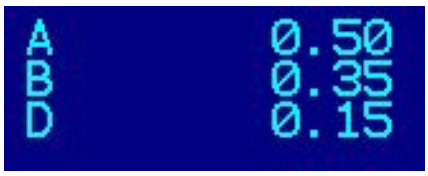


Рис. 2.3. Пример задания столбиком трёх символов и соответствующих им вероятностей

На рис. 2.3. Символу «A» соответствует вероятность появления 0,5. Символам «B» и «D» соответственно — 0,35 и 0,15.

Рис. 2.4. Пример задания трёх символов и соответствующих им вероятностей в отдельных строках

На рис. 2.4. Символу «+» соответствует вероятность появления 0,5. Символам «–» и «*» соответственно — 0,35 и 0,15.

Рис. 2.5. Пример задания трёх символов с использованием смещения и соответствующих им вероятностей в отдельных строках

На рис. 2.5. число 48 означает начальное смещение в таблице ASCII и все последующие вероятности будут соответствовать очередным символам из этой таблицы. Таким образом, символу «0» соответствует вероятность появления 0,5. Символам «1» и «2» соответственно — 0,35 и 0,15. сумма всех задаваемых для каждого символа вероятностей должна быть равна 1. Поэтому в программе должна быть проверка выполнения этого условия.

На рис. 6 приведена схема, которая поясняет как пользуясь имеющейся в языках программирования функцией, реализующей генератор псевдослучайной последовательности (ПСП), выполнить генерацию символов с заданными вероятностями ($P(\text{A})=0{,}1, P(\text{B})=0{,}1, P(\text{C})=0{,}1$). Параметр RAND_MAX — максимальное генерируемое функцией ПСП псевдослучайное число.

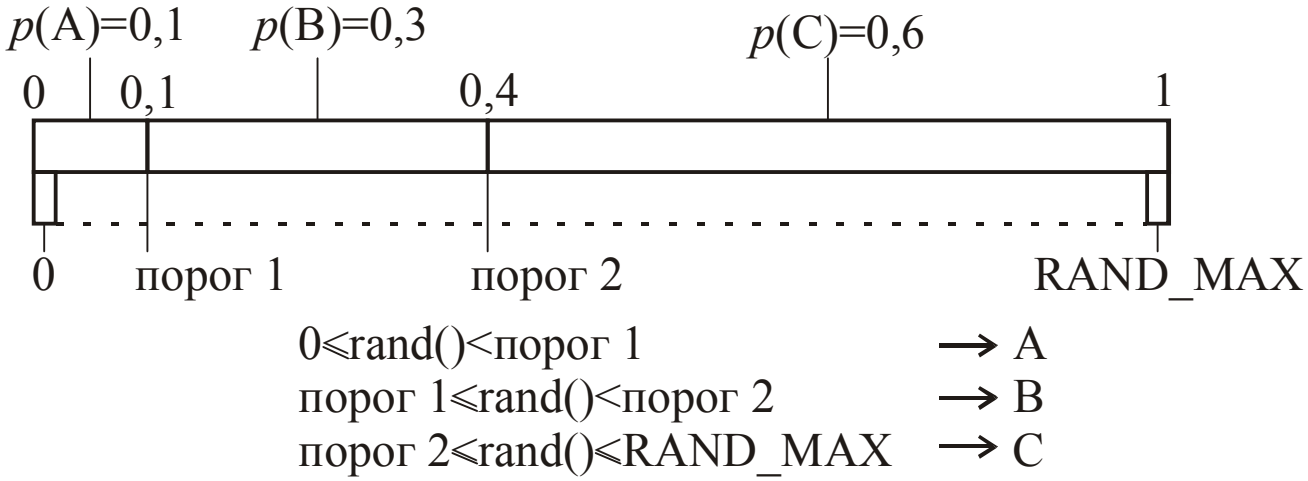


Рис. 2.6. Генерация независимых символов для $m=3$

Результат генерации должен быть сохранён в файле. Поэтому должна быть предусмотрена возможность задания имени выходного файла. Если вероятности будут заданы в текстовом файле, имеющим некоторое имя, то имя выходного файла можно автоматически назначать тем же, меняя только расширение файла.

Как вариант, по желанию, число генерируемых символов можно определять точностью задания вероятностей для символов (хотя бы одной из них).

После генерации результата необходимо отдельной программой выполнить проверку сгенерированных символов. Для этого проверяющая программа должна прочитать сгенерированный файл и подсчитать количество появлений уникальных символов. Делением количества уникальных символов на общее количество генерируемых, найти их частоты появлений.

Вывести в файл или в просмотровом окне вероятность генерации и частость появления для каждого символа так, чтобы можно было легко проверять соответствие частости вероятности. Показать и пояснить формат записи данных в файл. Рассчитать безусловную энтропию данного источника

$$H(X)=-\sum_i p(a_i)\log_2 p(a_i)$$

где $p(a_i)$ — вероятность появления символа a_i.

Задание 2.2

Реализовать программным способом генератор зависимых символов.

Генерацию текущих символов, в зависимости от предыдущих, можно пояснить рисунком 7.

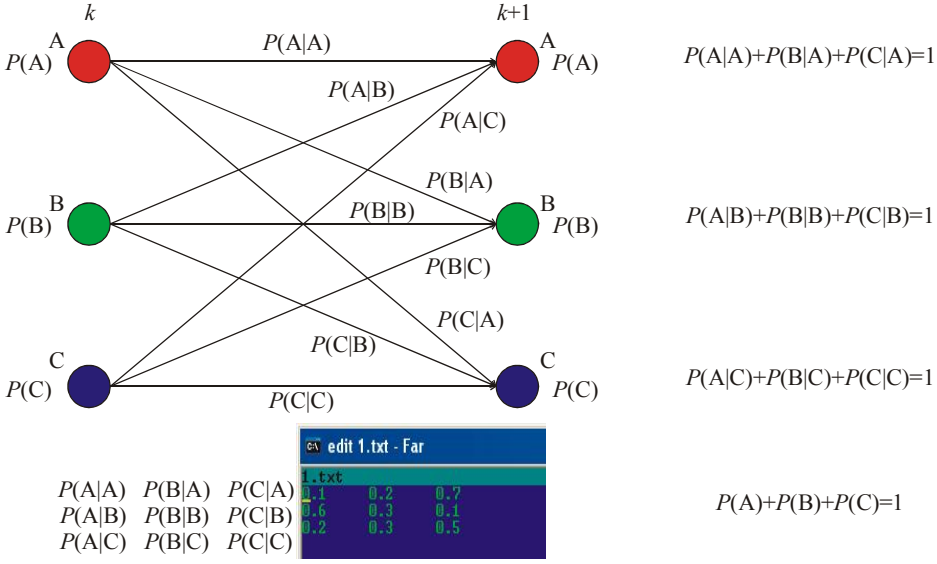


Рис. 2.7. Переходные вероятности от шага k к шагу $k+1$ при генерации $k+1$ символа

На рис. 2.7 проиллюстрирован процесс соответствия условных вероятностей при генерации символов «A», «B» и «C». Безусловные вероятности генерируемых символов соответственно равны $P(\mathrm{A})$, $P(\mathrm{B})$ и $P(\mathrm{C})$. Однако для генератора, в котором вероятность появления текущего символа зависит от предыдущего, достаточно задать только матрицу условных вероятностей (пример внизу на рис. 2.6). Например, условная вероятность $P(\mathrm{A}|\mathrm{B})$ означает вероятность появления символа «A» при условии, что предыдущим символом был «B».

Генерацию первого символа выполнить на основе безусловных вероятностей. При этом безусловные вероятности могут быть найдены путём решения системы линейных уравнений. Например, из уравнений

$$P(\mathrm{A})P(\mathrm{A}|\mathrm{A})+P(\mathrm{B})P(\mathrm{A}|\mathrm{B})+P(\mathrm{C})P(\mathrm{A}|\mathrm{C})=P(\mathrm{A});$$
$$P(\mathrm{A})P(\mathrm{B}|\mathrm{A})+P(\mathrm{B})P(\mathrm{B}|\mathrm{B})+P(\mathrm{C})P(\mathrm{B}|\mathrm{C})=P(\mathrm{B});$$
$$P(\mathrm{A})+P(\mathrm{B})+P(\mathrm{C})=1;$$

можнополучить систему вида:

$$\begin{cases} P(\mathrm{A})\big(1+P(\mathrm{A}|\mathrm{C})-P(\mathrm{A}|\mathrm{A})\big)+P(\mathrm{B})\big(P(\mathrm{A}|\mathrm{C})-P(\mathrm{A}|\mathrm{B})\big)=P(\mathrm{A}|\mathrm{C}) \\ P(\mathrm{A})\big(P(\mathrm{B}|\mathrm{C})-P(\mathrm{B}|\mathrm{A})\big)+P(\mathrm{B})\big(1+P(\mathrm{B}|\mathrm{C})-P(\mathrm{B}|\mathrm{B})\big)=P(\mathrm{B}|\mathrm{C}) \end{cases}$$

Генератор также должен быть рассчитан на произвольное количество генерируемых символов. Условные вероятности должны быть заданы в виде квадратной матрицы с обязательным отображением этой матрицы в тестовом файле.

Также как и для предыдущего варианта генератора должен быть параметр, определяющий количество генерируемых символов (объём выборки) и имя файла, в котором будет записан результат генерации.

Отдельной программой выполнить проверку сгенерированных данных. То есть, проверить условные и безусловные вероятности подсчётом условных и безусловных частостей и наглядного сопоставления этих значений.

Для заданных и найденных вероятностей найти безусловную и условную энтропию.

Условная энтропия определяется по формуле

$$H(A|A')=\sum_{j=1}^{K}H\left(A|a_j\right)P\left(a_j\right)=\sum_{i=1}^{K}\sum_{j=1}^{K}P\left(a_j\right)P\left(a_i|a_j\right)\log P\left(a_i|a_j\right)$$

где $p\left(a_i\right)$ — вероятность появления символа a_i, а $p\left(a_i|a_j\right)$ — вероятность появления символа a_i, при условии, что предыдущим был символ a_j.

На рис. 8 приведена схема работы генератора, в котором генерируются 3 варианта символов, согласно некоторой заданной матрице условных вероятностей. Цветами в прямоугольниках условно показаны диапазоны, по размеру соответствующие вероятности для каждого генерируемого символа. Верхние три прямоугольника характеризуют матрицу условных вероятностей, а нижний — расчётные значения безусловных вероятностей. Стрелками показан замкнутый процесс работы этого генератора.

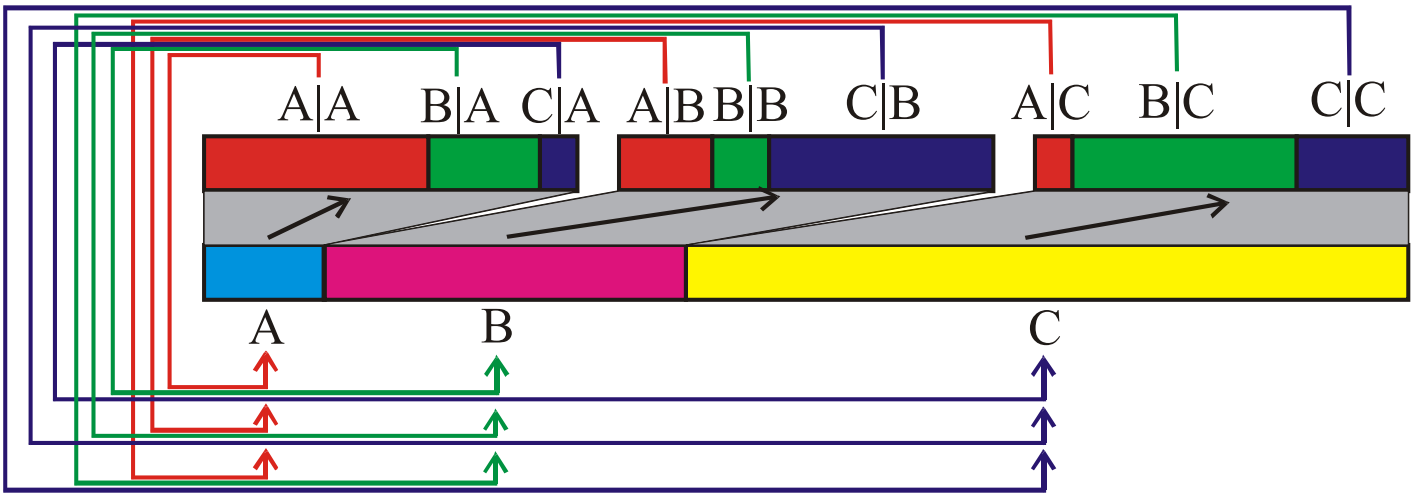


Рис. 2.8. Генерация зависимых символов для $m=3$

Таким образом, если реализован генератор по заданию 2.1, то генератор 2.2 представляет собой этап генерации предыдущего генератора с заменой вектора вероятностей генерируемых символов после каждого сгенерированного символа. Именно сгенерированный символ и будет определять выбор

вектора (строки из матрицы) для генерации следующего символа.

Для примера приведена программа на языке C++, которая реализует данный генератор и выполняет проверку и расчёт параметров сгенерированных данных. Вероятности переходов задаются в виде матрицы значений условной вероятности в текстовом файле.

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include <cstdlib>
using namespace std;
int main(int mn, char* nm[])
{ int i,j,k,l,row,max,C,N,t,K,Col,ms,s,*col; double q,x,Q_,X_,Q,X; char c,*p,*d,*b;
if(mn!=5) cerr<<"gen in.txt out.ext K N"<<endl,exit(1);
ifstream in(nm[1],ios::binary); if(!in) cerr<<"File\"."<<nm[1]<<"\"not open"<<endl,exit(1);
in.seekg(0,ios::end);s=in.tellg();in.seekg(0,ios::beg);p=new char[s+2];in.read((char*)p,s);in.close();
for(p[s++]=0x0A,p[s++]=0,i=-1;++i,p[i]+=23*!(p[i]-'\t');); //Последний символ 0x0A. Убираем табуляцию
for(i=-1;++i,p[i]+=2*!(p[i]-',');); //Заменяем запятые на точки
for(k=i=0;(p[k]=p[i]);i++) k+=(p[i]=='.'||(p[i]|7)=='7'||(p[i]|1)=='9'||p[i]==' '||p[i]==0x0A);
                                                    //оставить точки пробелы символы 0x0A и цифры от 0 до 9
for(ms=k=0,i=-1;p[++i];k+=!(p[i]-.')) if(p[i]==' '||p[i]==0x0A) (ms<k)?ms=k,k=0:k=0;
                                          // подсчёт максимального количества десятичных точек в одном слове (числе)
if(ms>1) cerr<<"в некоторых числовых значениях более одной десятичной точки\n",exit(1);
for(i=1;p[++i];) if(p[i-1]=='.') if(p[i-2]==' ') if(p[i]==' ') p[i-1]=' '; // замена одиночной точки на пробел
for(c=' ',k=i=0;(p[k]=p[i]);c=p[i++]) if(!(c^p[i])&&c==' '); else p[k++]=p[i];
                                                                    //Заменяет 2 и более пробелов на 1 пробел
for(i=-1; p[++i];) if((p[i+1]==0x0A&&p[i]==' ')||(p[i+1]==' '&&p[i]==0x0A)) p[i]=p[i+1]=0x0A;
                                                    //замена пробелов в начале и конце строк на символ 0x0A
for(c=0x0A, k=i=0; (p[k]=p[i]); c=p[i++]) if(!(c^p[i])&&(c==0x0A)); else p[k++]=p[i];
                                                        //Заменяет 2 и более символов 0x0A на 1 символ 0x0A
for(row=i=0;p[i];row+=(p[i++]==0x0A)); //Считает строки и записывает в переменную row
cout<<"row="<<row<<endl;
 {ofstream o("1.fb",ios::binary);for(i=-1;++i<s;o.put(p[i]));o.close(); }//проверка содержимого рабочего массива
col=new int[row]; for(i=row;--i>=0;col[i]=0);
                                          // массив для количества пробелов в каждой строке (на 1 меньше, чем чисел)
for(k=i=0;p[i];col[k]+=(p[i]==0x20),k+=(p[i++]==0x0A)); // подсчёт числа пробелов в строках и запись их в массив
for(s=i+1,k=i=-1;++i<row;k&=!(col[0]^col[i])); // проверка одинаковости количества пробелов
if(!k) cerr<<"В строках разное количество данных!\n",exit(3);
for(Col=col[0]-(i=-1); ++i<s-2;p[i]+=22*!(p[i]-0x0A)) if(!col[0])col[0]=row-1,row=1;
                                                    //замена символов 0x09 на пробел превращение столбика в строку
cout<<"s="<<s<<"\nCol="<<Col<<endl;
if(row-1) if(row!=Col)cerr<<"Матрица не квадратная!\n",exit(4);
 {ofstream o("2.fb",ios::binary);for(i=-1;++i<s;o.put(p[i]));o.close(); }//проверка содержимого рабочего массива
 { cout<<p; } //проверка содержимого рабочего массива
d=new char[s*10]; b=new char[s*10];//создание временного увеличенного массива
for(c=j=i=-1;p[++i];d[++j]=p[i],c=p[i])     if(p[i]=='.'&&c==' ') d[++j]=0x30;
 d[++j]=0; // добавление нуля перед десятичной точкой, если там был пробел
 { cout<<d<<endl; } //проверка содержимого рабочего массива
for(k=0,j=i=-1;d[++i];b[++j]=d[i]) if(k+=(d[i]=='.'),d[i]==' ') k?k=0:b[++j]='.';
                                                     // добавление десятичных точек в конце числа, если их там нет
if(!k) b[j]='.',b[j+1]=0x0A,b[j+2]=0; // добавление десятичной точки в конце последнего числа с сохранением 0x0A
 { cout<<endl<<b<<endl; } //проверка содержимого рабочего массива
for(max=k=0,i=-1;b[++i];)if(k*=(b[i]!='.'),b[i]==0x20||b[i]==0x0A){if(max<k) max=k; k=0;} else k++;
                                                  // подсчёт максимального числа знаков после запятой (увеличено на 1)
cout<<"max="<<max-1<<endl;
for(l=i=-1,k=0;b[++i];d[++l]=b[i],k*=!!j)
 if(k*=(b[i]!='.'),j=(b[i]!=0x20&&b[i]!=0x0A),k+=j,!j) for(;k++<max; d[++l]=48);
d[++l]=0;
 { cout<<d<<endl; } //проверка содержимого рабочего массива
for(j=i=-1;d[++i];) if(d[i]=='.'); else d[++j]=d[i];
d[++j]=0;
 //проверка содержимого рабочего массива
 { cout<<d<<endl; } //проверка содержимого рабочего массива
int*g=new int[row*(col[0]+1)]; int**G=new int*[row]; for(i=-1; ++i<row;) G[i]=&g[(col[0]+1)*i];
for(g[k=i=0]=atoi(&d[0]);++k<row*Col;)for(;d[++i]!=' ';) g[k]=atoi(&d[i+1]);
for(k=-1;d[++k]!=' ';);
for(i=l=1;++i<=k;l*=10);
cout<<"l="<<l<<endl;
for(i=-1;++i<row;cout<<endl) for(cout<<i<<": ",j=-1;++j<Col;) cout<<G[i][j]/float(l)<<'\t';
for(k=i=-1; ++i<row;s!=l?cerr<<"Сумма вероятностей в строке "<<i+(k=0)<<" не равно 1 "<<endl:cerr)
for(s=j=0;j<Col;) s+=G[i][j++];
if(!k)exit(6);
float *f = new float[row*Col]; float **F=new float*[row];  for(i=-1;++i<row;) F[i]=&f[Col*i];
for(i=-1;++i<row*Col;f[i]=g[i]/float(l));
for(i=-1;++i<row;cout<<endl) for(k=-1;++k<=col[i];) cout<<F[i][k]<<'\t';
for(K=2; K<l; K*=2);
cout<<"K="<<K<<endl;
for(i=-1;++i<row;)for(j=0;++j<=col[0];)G[i][j]+=G[i][j-1];
for(i = -1; ++i<row;cout<<endl) for(j=-1;++j<(col[0]+1);) cout<<G[i][j]<<'\t';
C=atoi(nm[3]);if(C<0||C-row>255)cerr<<" Ошибка диапазона\n",exit(7);
N=atoi(nm[4]);if(N<1)cerr<<"Ошибка N<1\n",exit(8);
srand(time(NULL));
```

```
if(row==1) // генератор по строке
 { ofstream ou(nm[2], ios::binary);
 for(i=0;i<N;)for(t=rand()%K, j=0;j<=col[0];j++)if(t<G[0][j])ou.put(C+j),j=col[0],i++; // генерация символов
 ou.close();
 for(q=i=0;i<=col[0];i++) if(F[0][i]>1e-9) q-=F[0][i]*log(F[0][i])/log(2.); //рассчёт безусловной энтропии
 x=1-q/(log(float(col[0]+1))/log(2.)); //рассчёт безусловной избыточности
 //вывод энтропии(сколько байт требуется на кодирование одного числа)
 cout<<"H(A)="<<q<<"\nx="<<x<<endl;
 cout<<"n/n0="<<1/(1-x)<<endl; //вывод коэффициента сжатия для независимых символов
 }
else //Генератор по матрице
 { int rw=row-1,idx; float w,mx,sm,ml;float*R=new float[row]; float*B=new float[rw]; float *P=new float[row+1];
 float*u=new float[rw*rw]; float **U = new float*[rw];for(i=-1;++i<rw;)U[i]=&u[rw*i];
 float*e=new float[rw*rw]; float **E = new float*[rw];for(i=-1;++i<rw;)E[i]=&e[rw*i];
 for(i=-1;++i<rw;B[i]=F[rw][i],cout<<F[rw][i]<<endl) // формирование уравнений системы
  for(j=0;j<rw;j++) U[i][j]=(i==j)+F[rw][i]-F[j][i],cout<<(i==j)<<"+"<<F[rw][i]<<"-"<<F[j][i]<<" ";
// Решение системы уравнений
 for(i=0;i<rw;i++) for(j=0;j<rw;j++) E[i][j]=(i==j);
 for(k=0;k<rw;k++)
  {
  for(mx=U[k][k],idx=k,i=k+1;i<rw;i++)
   if(fabs(U[i][k])>fabs(mx))mx=U[idx=i][k];
   for(i=k+1;i<rw;i++)
    if(U[k][k])
     for(ml=U[i][k]/U[k][k],j=0;j<rw;j++)U[i][j]-=ml*U[k][j],E[i][j]-=ml*E[k][j];
     else cerr<<"Err",exit(8);
  }
 for(k=rw-1;k>=0;k--)
  for(i=k-1;i>=0;i--)
   if(U[k][k]) for(ml=U[i][k]/U[k][k],j=0;j<rw;j++)E[i][j]-=ml*E[k][j]; else cerr<<"Err",exit(8);
 for(k=0;k<rw;k++) for(i=0;i<rw;i++) E[k][i]/=U[k][k];
 for(sm=i=0;i<rw;sm+=(P[i]=R[i]),i++) // получение безусловных вероятностей
   for(R[i]=j=0;j<rw;j++)
    R[i]+=E[i][j]*B[j];
 P[rw]=R[rw]=1-sm; // расчёт последней безусловной вероятности
 for(i=0;i<=rw;i++) cout<<"P("<<i<<")="<<P[i]<<endl; // вывод рассчитанных вероятностей
 for(i=1;i<=rw;i++) R[i]+=R[i-1]; // расчёт порогов для генерации символов (последний порог = 1 - не нужен)
 ofstream ou(nm[2], ios::binary);
 for(t=rand(),i=-1;++i<rw;) if(t<RAND_MAX*R[i]) ou.put(C+(k=i)),i=row; // генерация первого символа
 if(i<=row) k=rw,ou.put(c+k);
 for(i=1;i<N;) for(t=rand()%K,j=0;j<=rw;j++) if(t<G[k][j]) ou.put(C+(k=j)),j=row,i++;
                                                                  // генерация остальных символов
 for(Q=i=0;i<=row;i++) if(P[i]>1e-9)Q-=P[i]*log(P[i])/log(2.); //рассчёт безусловной энтропии
 X=1-Q/(log(float(row))/log(2.)); //рассчёт безусловной избыточности
 ou.close();
 for(Q_=i=0;i<row;Q_-=P[i++]*q) //рассчёт условной энтропии
  for(q=j=0;j<row;j++)if(F[i][j])
    q+=F[i][j]*log(F[i][j])/log(2.);
 X_=1-Q_/(log(float(row))/log(2.)); //рассчёт условной избыточности
 cout<<"H(A)= "<<Q<<"\n x(A)="<<X<<"\nn/n0="<<1/(1-X)<<endl;
           //вывод безусловной энтропии, безусловной избыточности, коэффициента сжатия для независимых символов
 cout<<"H(A|A^)="<<Q_<<"\nx(A|A^)="<<X_<<"\nn_/n0="<<1/(1-X_)<<endl;
                   //вывод условной энтропии, условной избыточности, коэффициента сжатия для зависимых символов
 delete[]R; delete[]B; delete[]U; delete[]E; delete[]u; delete[]e; delete[]P;
 }
delete[]g; delete[]G; delete[]d; delete[]p;
return 0;
}
```

Проверка частости появления символов

```
#include<iostream>
#include<fstream>
using namespace std;
int main(int mn,char* nm[])
{ int i,j,l,s,m; unsigned char* p;
if(mn!=2) cerr<<nm[0]<<" in.cb\n",exit(1);
ifstream in(nm[1],ios::binary); if(!in) cerr<<"Файл \""<<nm[1]<<"\" не открыт!\n",exit(2);
in.seekg(0,ios::end);s=in.tellg();in.seekg(0,ios::beg);
p=new unsigned char[s]; in.read((char*)p,s); in.close();
int* N=new int[65536]; int* k=new int[256]; int** n=new int*[256]; int* h=new int[s];
for(i=-1;++i<256;k[i]=0,n[i]=&N[i*256]);
for(i=-1;++i<65536;N[i]=0); for(i=-1;++i<s;h[i]=0);
for(i=-1;++i<s-1;N[*((short*)&p[i])]++,k[p[i]]++); k[p[i]]++;
for(m=0,i=-1;++i<256;m+=!!k[i]);
for(i=-1;++i<256;) for(l=j=-1;++j<256;) if(n[i][j]) h[++l]+=n[i][j];
for(i=-1;++i<256;) if(k[i]) cout<<"P("<<char(i)<<") = "<<k[i]/float(s)<<"\t("<<k[i]<<")\n";
for(i=-1;++i<256;)
 for(l=j=-1;++j<256;)
  if(n[i][j]) cout<<"P("<<char(i)<<"|"<<char(j)<<") = "<<n[i][j]/float(h[++l])<<"\t("<<n[i][j]<<")\n";
delete [] h; delete [] k; delete [] p; delete [] N; delete [] n; return 0;
}
```

Лабораторная работа №3. Экономное кодирование

Реализовать программным способом архиватор, сжимающий и восстанавливающий файлы, согласно заданному алгоритму. Архивация и деархивация может быть реализована либо одним и тем же программным исполняемым модулем, либо двумя отдельными. Исходная сживаемая информация получается путём генерации согласно заданию 2.1 и/или 2.2. Результат сжатия должен быть записан в бинарный файл. Расширение бинарного файла выбрать самостоятельно. Формат сжатого файла должен быть продемонстрирован и подробно пояснён. Для результирующего файла подсчитать среднюю длину в битах и результат представить сопоставляя его с расчётными значениями безусловной и условной энтропии из результата заданий 2.1 и 2.2. Также произвести сжатие этого же файла любым архиватором, которым часто используете. В результирующем файле определить размер сжатой информации, и также рассчитать её среднюю длину. Варианты алгоритмов сжатия и краткое их описание приведены ниже.

Код Хаффмена

Построение этого кода рассмотрим на конкретном примере источника с объёмом алфавита $K = 8$.

Символы (буквы) алфавита располагают в порядке убывающей вероятности, затем выбирают пару букв с наименьшими вероятностями (0,02 и 0,03), от них проводят прямые (ветви на кодовом дереве) до узла I — точки условного символа с суммарной вероятностью $0{,}02 + 0{,}03 = 0{,}05$. При этом ветви, проведённой сверху вниз, приписывают символ 1, а ветви, проведённой снизу вверх, — символ 0. Среди символов a_1, a_6 и I снова находят пару символов с наименьшими вероятностями (0,08 и 0,05) и от них проводят прямые до точки II — точки условного символа с суммарной вероятностью $0{,}08 + 0{,}05 = 0{,}13$. Этот процесс продолжается дальше, пока построение не замыкается к вершине, то есть формирование кодовой комбинации идёт слева направо. Мы получили кодовое дерево, которое позволяет нам написать кодовые комбинации для всех символов, начиная построение с вершины дерева. Полученный неравномерный код является префиксным.

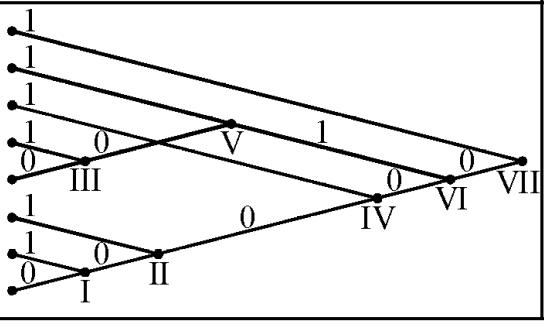
Символ источника	Вероятность $p(a_k)$	Кодовое дерево	Код	n_k	$n_k p(a_k)$
a_1	0,40		1	1	0,40
a_2	0,13		011	3	0,39
a_3	0,12		001	3	0,36
a_4	0,11		0101	4	0,44
a_5	0,11		0100	4	0,44
a_6	0,08		0001	4	0,32
a_7	0,03		00001	5	0,15
a_8	0,02		00000	5	0,10

Рис.3.1

Средняя длина $\overline{n}\ \dfrac{\text{бит}}{\text{символ}}$ кодовой комбинации в данном примере

$$\overline{n} = \sum_{k=0}^{7} n_k p_k = 2{,}6,$$

в то время как при примитивном кодировании двоичным кодом пришлось бы для всех кодовых комбинаций использовать три символа (разряда).

Предел Шеннона при двоичном коде для среднего числа символов на букву

$$\overline{n}_{\min} = \frac{H(A)}{\log_2 2} = H(A).$$

В нашем примере энтропия источника (считаем, что отсутствуют вероятностные связи между символами)

$$H(A) = -\sum_{k=1}^{8} p_k \log_2 p_k = 2{,}535\ \frac{\text{бит}}{\text{символ}}.$$

То есть код Хаффмена позволил получить близкое к теоретическому пределу значение средней длины кодового слова.

Программа кодера по Хаффмену

```
struct haff
{ unsigned s; int i; int p; char c; char k;int j;
void print() {cout<<"c="<<c<<" i="<<i<<" j="<<j<<" p="<<p<<" s="<<s<<" k="<<k<<endl;}
};
#define Q 0 // 1 - по вероятности, 0 - согласно ссылке на сортировку
int main(int mn,char* nm[])
{ int N,size,i,j,k,min,jmin1,jmin2,max,imax,Max=-2u/2,s,v;
int b[256]={0};
haff* h; short* c; unsigned char *p,g;
ExeFile(nm[0]); if(mn!=3) cerr<<nm[0]<<" in.ext out.arh\n",exit(1);
ifstream in(nm[1],ios::binary); if(!in) cerr<<"file \""<<nm[1]<<"\" not open!\n",exit(1);
in.seekg(0,ios::end);size=in.tellg();in.seekg(0,ios::beg);
p=new unsigned char[size];if(!p) cerr<<"Error memory!\n",exit(1);
in.read((char*)p,size);in.close();
for(i=0;i<size;i++) b[p[i]]++;
for(N=i=0;i<256;i++) N+=!!b[i];
cout<<"N="<<N<<endl;
h=new haff[2*N+1];if(!h) cerr<<"Error memory!\n",exit(1);
for(j=0;j<2*N-1;j++) h[j].c=h[j].s=h[j].i=h[j].k=h[j].p=h[j].j=0;
for(j=i=0;i<256;i++) if(b[i]) h[j].s=b[i],h[j++].c=i;
vector<vector<char>*> V(N);
for(j=-N;j<0;imax>=0?h[imax].i=h[imax].j=j++:j++) for(imax=-1,max=k=0;k<N;k++) if(!h[k].i)
if(h[k].s>max) max=h[k].s,imax=k;
for(s=j=0;j<N;s+=h[j++].s);
cout<<"s="<<s<<endl;
for(i=0;i<N-1;i++)
 { for(min=Max,jmin1=-1,j=0;j<N+i;j++) if(!h[j].p) if(min>=h[j].s) min=h[j].s,jmin1=j;
 h[jmin1].p=N+i;
 for(min=Max,jmin2=-1,j=0;j<N+i;j++) if(!h[j].p) if(min>=h[j].s) min=h[j].s,jmin2=j;
 h[jmin2].p=N+i;
 if(Q) h[jmin1].k=0x30,h[jmin2].k=0x31; // установка цифрового кода (0 или 1) по
вероятности
 else if(h[jmin1].j<h[jmin2].j) h[N+i].j=h[jmin1].j,h[jmin1].k=0x31,h[jmin2].k=0x30; else
h[N+i].j=h[jmin2].j,h[jmin1].k=0x30,h[jmin2].k=0x31;
 h[N+i].s=h[jmin1].s+h[jmin2].s;
 h[N+i].i=i+1;
 }
for(j=0;j<N;j++)
 { for(i=0,k=j;h[k].p;k=h[k].p,i++);
 V[j]=new vector<char>(i); if(!V[j]) cerr<<"Error memory!\n",exit(1);
 for(i=0,k=j;h[k].p;k=h[k].p,i++) (*V[j])[V[j]->size()-1-i]=h[k].k;
 }
for(j=0;j<N;j++,cout<<endl)
 for(k=(int)(unsigned char)h[j].c-32,k<0?cout<<"\'\\"<<k+32<<"\':
":cout<<"\'"<<char(k+32)<<"\': ",i=0;i<V[j]->size();i++) cout<<(*V[j])[i];
ofstream ou(nm[2],ios::binary); if(!ou) cerr<<"Error create the file
\""<<nm[1]<<"\"\n",exit(1);
g=N-1; ou.put(g);
for(j=0;j<N;j++)
 for(ou.put(h[j].c),g=V[j]->size(),ou.put(g),i=0;i<V[j]->size();i+=8, ou.put(g) )
for(g=k=0;k<8;++k) g|=((*V[j])[(i+k)%V[j]->size()]&1)<<(7-k);
for(i=0;i<256;b[i++]=-1); for(i=0;i<N;b[(unsigned char)h[i].c]=i++);
for(g=k=j=0;j<s;j++) for(i=0;i<V[b[p[j]]]->size();g|=((*V[b[p[j]]])[i++]&1)<<(7-
k%8),++k%8?1:(ou.put(g),g=0));
if(k%8) ou.put(g); g=k%8,ou.put(g); cout<<"Кодированная часть файла в битах: "<<k<<endl;
ou.close(); for(j=0;j<N;j++) delete V[j]; delete [] p; delete [] h; return 0;
}
```

Адаптивное кодирование по Хаффмену

Предлагается однопроходный алгоритм сжатия без передачи таблицы кодов.

При каждом сопоставлении символу кода, в следующем ходе вычислений этому же символу может быть сопоставлен другой код, то есть происходит адаптация алгоритма к поступающим для кодирования символам.

Дерево кодирования содержит вначале только один специальный символ, имеющий частоту 0. Он необходим для занесения в дерево новых символов.

Символы кодируются 8-битым кодом из расширенной таблицы ASCII.

При построении кода упорядочивается структура дерева. Листья дерева располагаются в порядке возрастания частот и затем в порядке возрастания стандартных кодов символов. Левые ветви помечаются 0, а правые — 1.

Бинарное дерево называется упорядоченным, если его узлы могут быть перечислены в порядке неубывания веса и узлы, имеющие общего родителя, должны находится рядом, на одном ярусе (рис. 3.2). Перечисление идёт по ярусам снизу-вверх и слева-направо в каждом ярусе.

Если дерево кодирования упорядоченно, то при изменении веса существующего узла дерево не нужно целиком перестраивать — в нём достаточно лишь поменять местами два узла: узел, вес которого нарушил упорядоченность, и последний из следующих за ним узлов меньшего веса. После перемены мест узлов, необходимо пересчитать веса всех узлов-предков.

Если на дереве добавить ещё две буквы A, то узлы A и D меняются местами (рис. 3.3)

Упорядоченное дерево

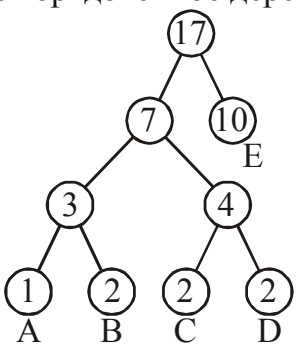


Рис. 3.2

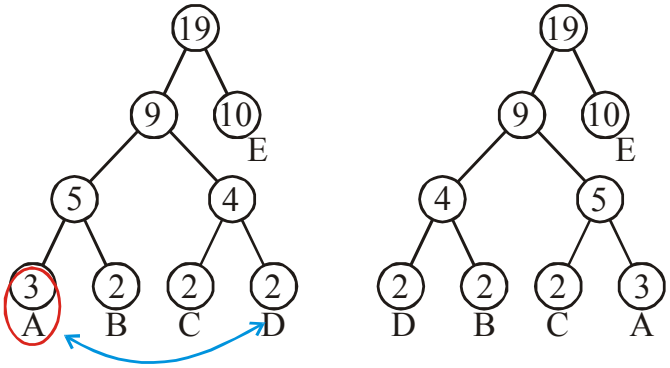


Рис. 3.3

Если добавить ещё две буквы A, то меняются местами сначала узел A и узел, родительский узлов D и B, а затем узел E и узел-брат E (рис. 3.4).

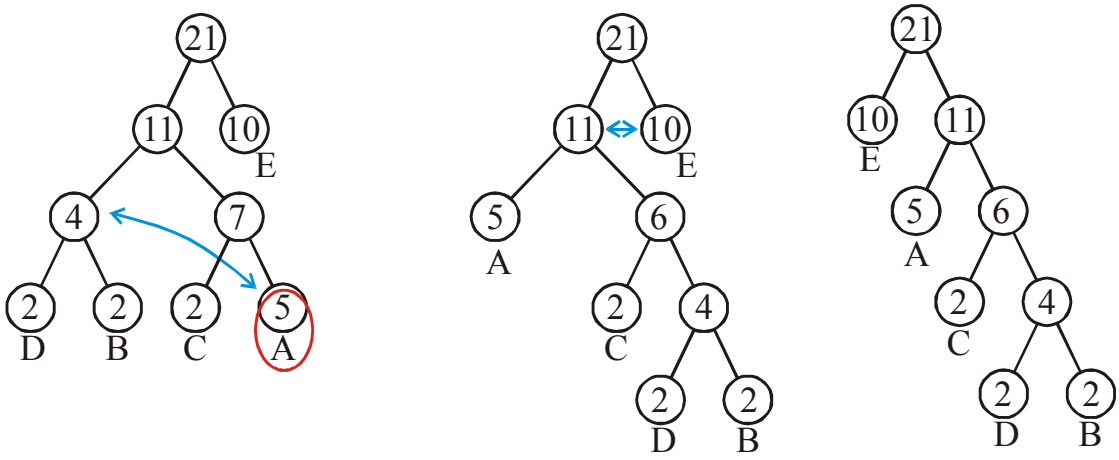
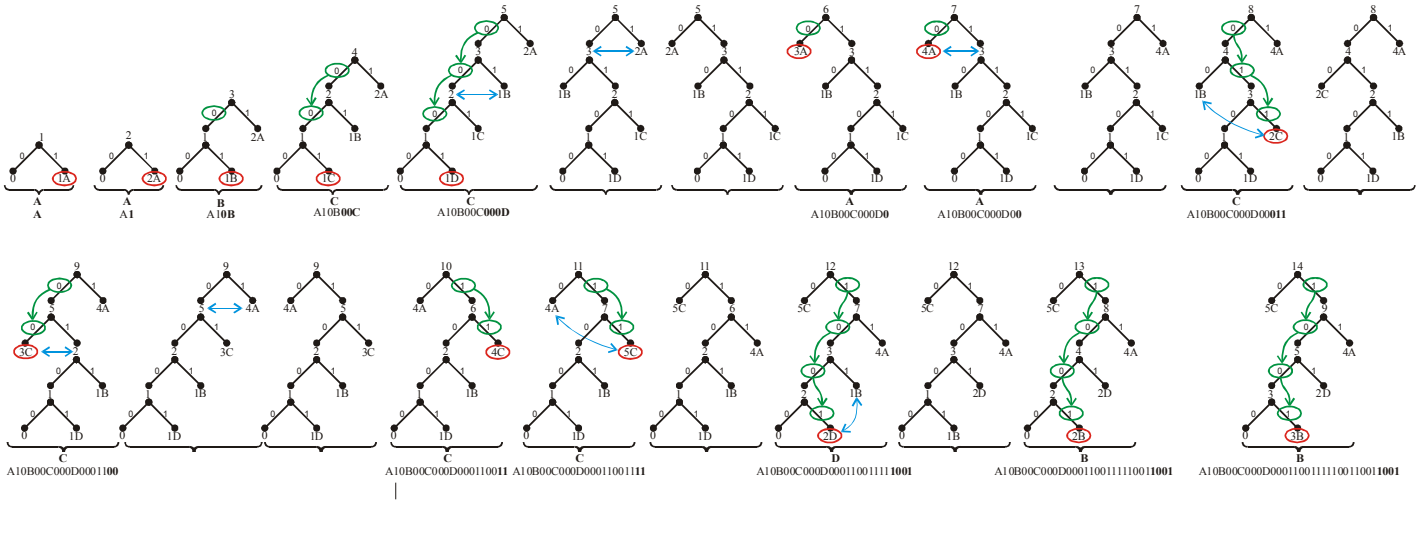


Рис. 3.4

На рис. 3.5 и 3.6 приведён процесс кодирования по адаптивному алгоритму Хаффмена (на рис. 3.5 для фразы «AABCDAACCCCDBB»>, а на рис 3.6 — для фразы «КИБЕРНЕТИКИ». Эти пояснения помогут при отладке данного алгоритма.

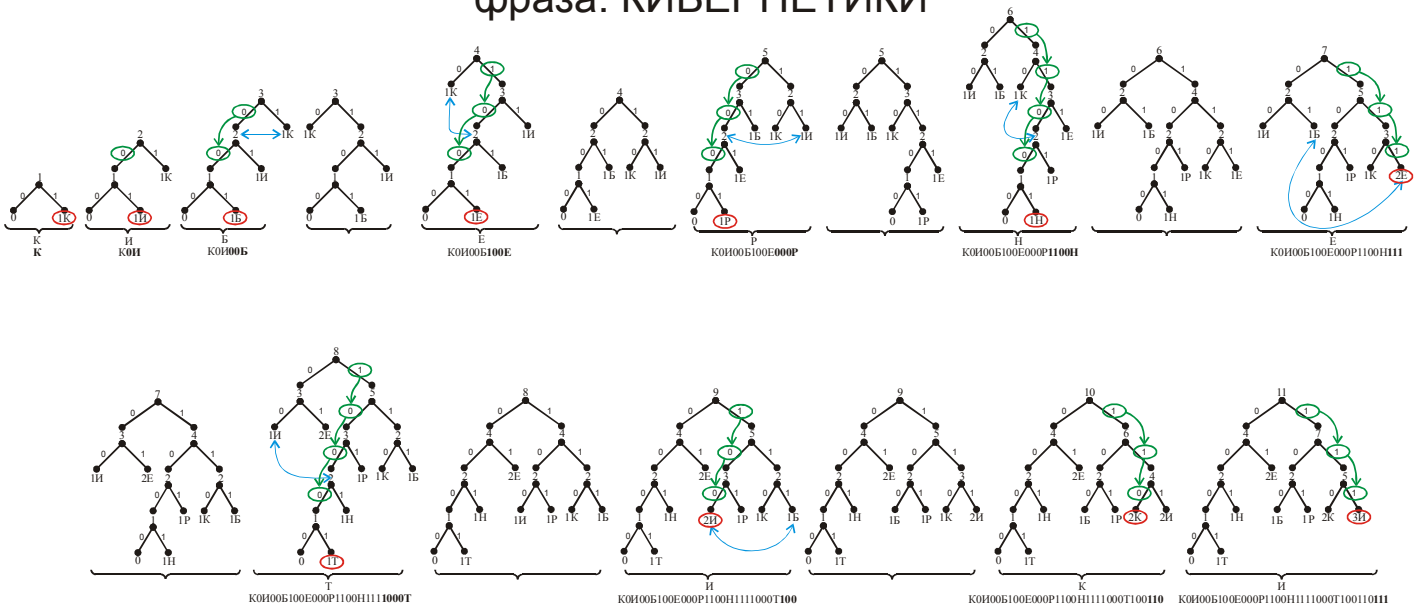
Кодирование по адаптивному алгоритму Хаффмана
фраза: AABCDAACCCCDBB



code(**AABCDAACCCCDBB**)=**A**10**B**00**C**000**D**0001100111110011001001=**01000001**10**01000010**00**01000011**000**01000100**0000110011111001001001
L(code(**AABCDAACCCCDBB**))=62 **бита**

Рис. 3.5

Кодирование по адаптивному алгоритму Хаффмана
фраза: КИБЕРНЕТИКИ



code(**КИБЕРНЕТИКИ**)=К0**И**00**Б**100**Е**000**Р**1100**Н**1111000**Т**10011011=**10001010**10**0010000**10**0000001**1001**0000101**0001**00100001**1001**0001101**11111000**10**010010**100110111
=**11001010**01**1001000**001**1000001**1001**1000101**0001**10100001**1001**1001101**11111000**11**010010**100110111
$_{\varphi 465}$ = $_{\varphi 1231}$
L(code(**КИБЕРНЕТИКИ**))=85 **бит**

Рис. 3.6

Код Шеннона Фэно

Процедура сводится к тому, что поэтапно, начиная от вершины, массив символов источника делится на две группы с примерно равными суммарными вероятностями. Символам первой группы приписывается символ 1, символам второй группы приписывается кодовый символ 0. Формирование кодовой комбинации идёт справа налево В нашем примере $\overline{n} = 2{,}65$, что на много больше, чем для кода Хаффмана. Средняя длина $\overline{n}\ \dfrac{\text{бит}}{\text{символ}}$ кодовой комбинации в данном примере $\overline{n} = \sum_{k=0}^{7} n_k p_k = 2{,}65$

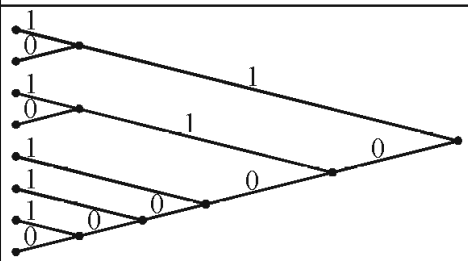
Символ источника	Вероятность $p(a_k)$	Кодовое дерево	Код	n_k	$n_k p(a_k)$
a_1	0,40		11	2	0,80
a_2	0,13		10	2	0,26
a_3	0,12		011	3	0,36
a_4	0,11		010	3	0,33
a_5	0,11		001	3	0,33
a_6	0,08		0001	4	0,32
a_7	0,03		00001	5	0,15
a_8	0,02		00000	5	0,10

Рис. 3.7

Если помимо неравномерного распределения символов первичного источника между ними также присутствует статистическая связь (соседние символы или группы символов зависимы), то рассмотренные коды Хаффмана и Шеннона-Фэно окажутся менее эффективными. Так, например, для русского языка избыточность, обусловленная неравновероятностью символов $\rho_{вер} = 0{,}13$, а статистической связью — $\rho_{св} = 0{,}73$. В этом случае более эффективным окажется подход, при котором описанными выше методами кодируются не отдельные символы источника, а целые последовательности символов. Такое кодирование называют *кодированием с укрупнением алфавита*. При глубоких статистических связях длины кодируемых цепочек значительно увеличиваются и возрастает сложность алгоритмов кодирования и декодирования, а также время, затрачиваемое на обработку.

Программа кодера по Шеннону Фэно

```
void haf(int a,int b,char k,int u,int& z,haff* h)
{ if(a==b) return;
int r,s,j,w; float t;
h[--z].p=u; h[z].k=k+0x30;
for(s=0,j=a;j<=b;j++) h[j].p=z,s+=h[j].s;
for(t=s/2.,w=0,j=a-1;w<t;w+=h[++j].s);
r=j-1+(t-w+h[j].s>w-t);
for(j=a;j<=b;h[j].k=(j<=r)+0x30,j++);
u=z; haf(a,r,1,u,z,h); haf(r+1,b,0,u,z,h);
}
int main(int mn,char* nm[])
{ int b[256]={0},N,size,i,j,k,max,imax,s,r,z,w,v;
float t; haff* h; unsigned char *p,g;
ExeFile(nm[0]); if(mn!=3) cerr<<nm[0]<<" in.ext out.arh\n",exit(1);
ifstream in(nm[1],ios::binary);
if(!in) cerr<<"file \""<<nm[1]<<"\" not open!\n",exit(1);
in.seekg(0,ios::end);size=in.tellg();in.seekg(0,ios::beg);
p=new unsigned char[size];if(!p) cerr<<"Error memory!\n",exit(1);
in.read((char*)p,size);in.close();
for(i=0;i<size;i++) b[p[i]]++;
for(N=i=0;i<256;i++) N+=!!b[i];
cout<<"Размер таблицы равен "<<N<<" символам."<<endl;
h=new haff[2*N+1];if(!h) cerr<<"Error memory!\n",exit(1);
for(j=0;j<2*N-1;j++) h[j].c=h[j].s=h[j].i=h[j].k=h[j].p=h[j].j=0;
for(j=i=0;i<256;i++) if(b[i]) h[j].s=b[i],h[j++].c=i;
vector<vector<char>*> V(N);
for(j=-N;j<0;imax>=0?h[imax].i=h[imax].j=j++:j++)
 for(imax=-1,max=k=0;k<N;k++) if(!h[k].i) if(h[k].s>max) max=h[k].s,imax=k;
for(s=j=0;j<N;s+=h[j++].s);
cout<<"Количество символов в исходном файле равно "<<s<<".\n";
z=2*N-2;
for(s=0,j=0;j<N;j++) h[j].p=z,s+=h[j].s;
for(t=s/2.,w=0,j=-1;w<t;w+=h[++j].s);
r=j-1+(t-w+h[j].s>w-t);
for(j=0;j<N;h[j++].k=(j<=r)+0x30);
haf(0,r,1,2*N-2,z,h); haf(r+1,N-1,0,2*N-2,z,h);
for(j=0;j<N;j++)
 { for(i=0,k=j;h[k].p;k=h[k].p,i++);
 V[j]=new vector<char>(i); if(!V[j]) cerr<<"Error memory!\n",exit(1);
 for(i=0,k=j;h[k].p;k=h[k].p,i++) (*V[j])[V[j]->size()-1-i]=h[k].k;
 }
```

```
for(j=0;j<N;j++,cout<<endl)
 for(k=(int)(unsigned char)h[j].c-32,k<0?cout<<"\'\\"<<k+32<<"\':
":cout<<"\'"<<char(k+32)<<"\': ",i=0;i<V[j]->size();i++) cout<<(*V[j])[i];
ofstream ou(nm[2],ios::binary); if(!ou) cerr<<"Error create the file
\""<<nm[1]<<"\"\n",exit(1);
g=N-1; ou.put(g);
for(j=0;j<N;j++)
 for(ou.put(h[j].c),g=V[j]->size(),ou.put(g),i=0;i<V[j]->size();i+=8, ou.put(g) )
for(g=k=0;k<8;++k) g|=((*V[j])[(i+k)%V[j]->size()]&1)<<(7-k);
for(i=0;i<256;b[i++]=-1); for(i=0;i<N;b[(unsigned char)h[i].c]=i++);
for(g=k=j=0;j<s;j++) for(i=0;i<V[b[p[j]]]->size();g|=((*V[b[p[j]]])[i++]&1)<<(7-
k%8),++k%8?1:(ou.put(g),g=0));
if(k%8) ou.put(g); g=k%8,ou.put(g); cout<<"Кодированная часть файла в битах: "<<k<<endl;
ou.close(); for(j=0;j<N;j++) delete V[j]; delete [] p; delete [] h; return 0;
}
```

Программа декодера файлов, закодированных по Хаффмену или Шеннону Фэно

```
int main(int mn,char* nm[])
{ int o=0,N,size,i,j,k,l=0,r,e,t,ii;
unsigned char *p,q; short *c,v; char* a;
ExeFile(nm[0]); if(mn!=3) cerr<<nm[0]<<" in.arh out.ext\n",exit(1);
ifstream in(nm[1],ios::binary); if(!in) cerr<<"file \""<<nm[1]<<"\" not open!\n",exit(1);
in.seekg(0,ios::end);size=in.tellg();in.seekg(0,ios::beg);
p=new unsigned char[size];if(!p) cerr<<"Error memory!\n",exit(1);
in.read((char*)p,size);in.close();
N=(unsigned char)p[l++]+1;
a=new char[N]; c=new short[2*N-2];
for(i=0;i<2*N-2;c[i++]=-1);
cout<<"Размер таблицы равен "<<N<<" символам."<<endl;
for(j=0;j<N;j++,cout<<endl)
 {
 a[j]=p[l++],r=(unsigned char)p[l++];
 k=(unsigned char)a[j]-32,k<0?cout<<"\'\\"<<k+32<<"\': ":cout<<"\'"<<char(k+32)<<"\': ";
 k=2*N-2; // начинаем с последней ячейки
 for(q=i=0;i<r;i++,q>>=1)
  {
  if(!q) t=(unsigned char)p[l++],q=128;
  v=!!(t&q),cout<<v;
  if(c[(k-N)*2+v]<0)
   if(i==r-1)
    c[(k-N)*2+v]=j;
   else
    {
    for(e=k-1;e>=N;--e) if(c[(e-N)*2+v]<0 && c[(e-N)*2+!v]<0) break;
    if(c[(e-N)*2+!v]<0)
     k=c[(k-N)*2+v]=e;
    else
     {
     for(;e>=N;--e) if(c[(e-N)*2+!v]<0 && c[(e-N)*2+v]<0) break;
     k=c[(k-N)*2+v]=e;
     }
    }
  else k=c[(k-N)*2+v];
  }
 }
cout<<"Заголовок "<<l<<" байт"<<endl;
size=(size-l-1-!!p[size-1])*8+p[size-1];
cout<<"Сжатый текст содержал "<<size<<" бит."<<endl;
ofstream ou(nm[2],ios::binary);
if(!ou) cerr<<"Error create the file \""<<nm[1]<<"\"\n",exit(1);
for(q=128,v=N-2,j=0;j<size;j++)
 { if(!q) q=128,l++;
 k=!!(p[l]&q),q>>=1;
 if(c[2*v+k]<N) ou.put(a[c[2*v+k]]),v=N-2;
 else v=c[2*v+k]-N;
 }
ou.close(); delete [] p, delete [] c, delete [] a; return 0;
}
```

Арифметическое кодирование

Алгоритм кодирования Хаффмана не может передавать на каждый символ сообщения менее одного бита информации. Схема кодирования, при которой некоторые символы кодируются менее, чем одним битом является арифметическое кодирование.

По исходному распределению вероятностей при выбранной для кодирования дискретной случайной величине строится таблица, состоящая из пересекающихся только в граничных точках отрезков для каждого из значений этой дискретной случайной величины. Объединение этих отрезков должно образовывать отрезок $[0,\,1]$, а их длины должны быть пропорциональными вероятностям соответствующих значений дискретных случайных величин.

Алгоритм кодирования заключается в построении отрезка, однозначно определяющего данную последовательность значений дискретных случайных величин. Затем для построенного отрезка находится число, принадлежащее его внутренней части и равное целому числу, делённому на минимально возможную положительную целую степень двойки. Это число и будет кодом для рассматриваемой последовательности. Все возможные конкретные коды — это числа строго большие нуля и строго меньшие одного, поэтому можно отбрасывать лидирующий ноль и десятичную запятую, но нужен ещё специальный код-маркер, сигнализирующий о конце сообщения. Отрезки строятся так. Если имеется отрезок для сообщения длины $n-1$, то для построения отрезка для сообщения длины *n*, разбиваем его на столько же частей, сколько значений имеет рассматриваемая дискретная случайная величина. Это разбиение делается совершенно также как и самое первое (с сохранением порядка). Затем выбирается из полученных отрезков тот, который соответствует заданной конкретной последовательности длины *n*.

Принципиальное отличие этого кодирования от рассмотренных ранее методов в его непрерывности, то есть в ненужности блокирования. Эффективность арифметического кодирования растёт с ростом длины сжимаемого сообщения (для кодирования Хаффмана и Шеннона-Фано этого не происходит). Недостатком арифметического кодирования является большие требования к вычислительным ресурсам.

При сжатии заданных данных, например, из файла все рассмотренные методы требуют двух проходов. Первый для сбора частот символов, используемых как приближённые значения вероятностей символов, и второй для собственно сжатия.

Пример. Пусть дискретная случайная величина *X* может принимать только два значения 0 и 1 с вероятностями $\frac{2}{3}$ и $\frac{1}{3}$ соответственно. Сопоставим значению 0 отрезок $\left[0,\frac{2}{3}\right]$, а 1 — $\left[\frac{2}{3},1\right]$. Тогда $\dim\left(\overrightarrow{X}\right)=3,\quad H(X)=H\left(\overrightarrow{X}\right)/3=\log_2 3-\frac{2}{3}\approx 0{,}9183\,\frac{\text{бит}}{\text{симв.}}$. Таблица построения кодов:

Интервалы и коды			Вероятность	Код Хаффмена
		$111[\frac{26}{27},1]\ni\frac{31}{32}=0.11111$	$1/27$	0000
	$11[\frac{8}{9},1]$	$110[\frac{8}{9},\frac{26}{27}]\ni\frac{15}{16}=0.1111$	$2/27$	0001
		$101[\frac{22}{27},\frac{8}{9}]\ni\frac{7}{8}=0.111$	$2/27$	010
$1[\frac{2}{3},1]$	$10[\frac{2}{3},\frac{8}{9}]$	$100[\frac{2}{3},\frac{22}{27}]\ni\frac{3}{4}=0.11$	$4/27$	001
		$011[\frac{16}{27},\frac{2}{3}]\ni\frac{5}{8}=0.101$	$2/27$	011
	$01[\frac{4}{9},\frac{2}{3}]$	$010[\frac{4}{9},\frac{16}{27}]\ni\frac{1}{2}=0.1$	$4/27$	100
		$001[\frac{8}{27},\frac{4}{9}]\ni\frac{3}{8}=0.011$	$4/27$	101
$0[0,\frac{2}{3}]$	$00[0,\frac{4}{9}]$	$000[0,\frac{8}{27}]\ni\frac{1}{4}=0.01$	$8/27$	11.

Рис. 3.8

$M\left(L_1\left(\overrightarrow{X}\right)\right)=\frac{65}{81}\approx 0{,}8025\,\frac{\text{бит}}{\text{симв.}}$ (арифметическое),

$M\left(L_1\left(\overrightarrow{X}\right)\right)=\frac{76}{81}\approx 0{,}9383\,\frac{\text{бит}}{\text{симв.}}$ (блочный Хаффмена),

$M\left(L_1\left(\overrightarrow{X}\right)\right)=M(L(X))=1\,\frac{\text{бит}}{\text{симв.}}$ (Хаффмена).

Среднее количество бит на единицу сообщения для арифметического кодирования получилось меньше, чем энтропия. Это связано с тем, что в рассмотренной простейшей схеме кодирования, не описан код-маркер конца сообщения, введение которого неминуемо сделает это среднее количество бит большим энтропии.

Получение исходного сообщения из его арифметического происходит по следующему алгоритму.

Шаг 1. В таблице для кодирования значений дискретной случайной величины определяется интервал, содержащий текущий код, — по этому интервалу однозначно определяется один символ исходного сообщения. Если этот символ — маркер конца сообщения, то конец.

Шаг 2. Из текущего кода вычитается нижняя граница содержащего его интервала. Полученное число считывается новым текущим значением кода. Переход к шагу 1.

Адаптивное арифметическое кодирование

Каждому символу сопоставляется его вес: вначале он для всех равен 1. Все символы располагаются в естественном порядке, например, по возрастанию. Вероятность каждого символа устанавливается равной его весу, делённому на суммарный вес всех символов. После получения очередного символа и постройки интервала для него, вес этого символа увеличивается на 1.

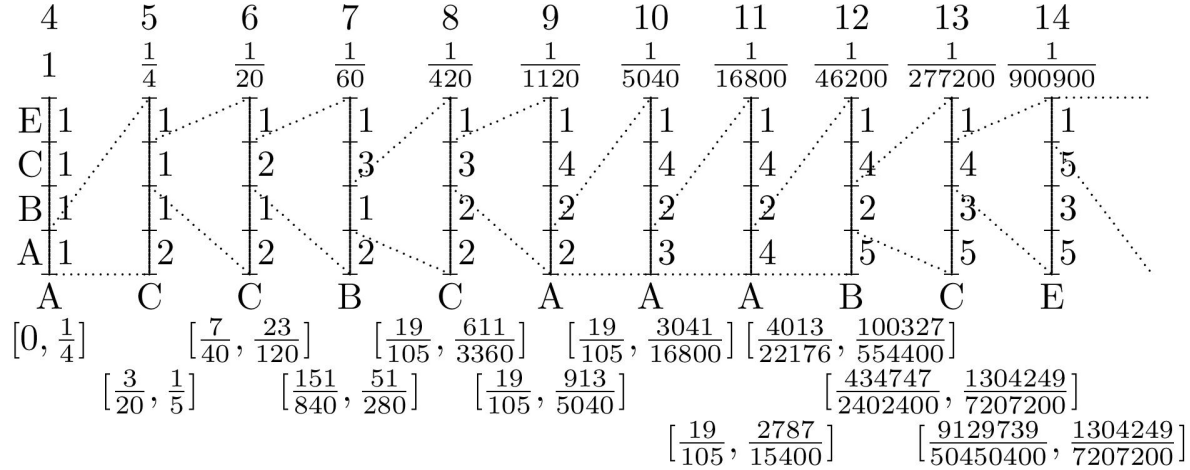


Рис. 3.9

Программа арифметического кодера (без описания функций работы со строками)

```
int main(int mn,char* nm[])
{ int i,j,k,size,b[256],y[256],sz,s; unsigned char *p,*r; if(mn==2||mn==3); else cerr<<"drob.exe in.ext
<tabl_kol.txt>\n",exit(1);
ifstream in(nm[1],ios::binary); if(!in) cerr<<"Not open \""<<nm[1]<<"\" file!\n",exit(1);
in.seekg(0,ios::end); size=in.tellg(); in.seekg(0,ios::beg); p=new unsigned char[size]; if(!p) cerr<<"No
memory!\n",exit(1);
in.read((char*)p,size); in.close(); memset(b,0,sizeof(int)*256); memset(y,-1,sizeof(int)*256);
if(mn==3) { char c; ifstream in(nm[2]); for(sz=0;!in.eof();sz++) in>>c>>i; in.close(); }
else { for(i=0;i<size;i++) b[p[i]]++; for(sz=i=0;i<256;sz+=!!b[i++]); }
cout<<sz<<endl; r=new unsigned char[sz];
if(mn==3)
 { int y; ifstream in(nm[2]); for(s=i=0;i<sz;i++) in>>r[i]>>y,b[r[i]]=y; in.close(); for(i=0;i<sz;i++)
cout<<unsigned(r[i])<<": "<<b[r[i]]<<endl; }
else
 { for(j=i=0;i<256;i++) if(b[i]) r[j++]=i; for(i=0;i<sz;i++) cout<<unsigned(r[i])<<": "<<b[r[i]]<<endl;
 for(s=i=0;i<sz;i++) s+=b[r[i]]; cout<<"s="<<s<<" size="<<size<<endl; }
vector<string> w(sz+1);
string z,lg,rg,zn,lg2,qq; { stringstream out; out<<size; z=out.str(); }
for(w[0]="0",s=0,i=1;i<=sz;i++) { s+=b[r[i-1]]; stringstream out;  out<<s; w[i]=out.str(); }
if(mn==3) { stringstream out;  out<<w[sz]; z=out.str(); }
for(i=0;i<sz;i++) y[r[i]]=i;
for(i=0;i<sz;i++) (r[i]<32?cout<<"\'\\"<<int(r[i]):cout<<"\'"<<r[i]),cout<<"\':
"<<w[i]<<"/"<<z<<"..."<<w[i+1]<<"/"<<z<<endl;
lg=w[y[p[0]]]; rg=w[y[p[0]]+1]; zn=z;
r[y[p[0]]]<32?cout<<"\'\\"<<int(r[y[p[0]]]):cout<<"\'"<<r[y[p[0]]]; cout<<"\':
"<<lg<<"/"<<zn<<"..."<<rg<<"/"<<zn<<endl;
for(i=1;i<size;i++)
 { lg2=add(mulp(lg,z),mulp(w[y[p[i]]],sub(rg,lg))); zn=mulp(zn,z);
rg=add(mulp(lg,z),mulp(w[y[p[i]]+1],sub(rg,lg))); lg=lg2;
 r[y[p[i]]]<32?cout<<"\'\\"<<int(r[y[p[i]]]):cout<<"\'"<<r[y[p[i]]]; cout<<"\':
"<<lg<<"/"<<zn<<"..."<<rg<<"/"<<zn<<endl;
 }
string t,tlg,trg,tzn,tch,ch,zz,s1,s2;
for(i=0;i<size;i++) cout<<p[i]; cout<<endl; for(i=0;i<sz;i++) cout<<r[i]; cout<<endl;
for(i=0;i<sz;i++) (r[i]<32?cout<<"\'\\"<<int(r[i]):cout<<"\'"<<r[i]),cout<<"\':
"<<w[i]<<"/"<<z<<"..."<<w[i+1]<<"/"<<z<<endl;
vector<int> W(256,-1); for(i=0;i<sz;i++) W[r[i]]=1; { stringstream out; out<<sz; z=out.str(); }
for(lg2=z,t="1",tlg="1",k=0;k<W.size();k++)
 if(W[k]>0) k<32?cout<<"\'\\"<<int(k):cout<<"\'"<<char(k),cout<<"\': "<<W[k]<<" ";
for(s=i=0;r[i]!=p[0];s+=W[r[i++]]); { stringstream out; out<<s; lg=out.str(); }
```

```
rg=addp(lg,mulp(tlg,t));
(p[0]<32?cout<<"\'\\"<<int(p[0]):cout<<"\'"<<p[0]),cout<<"\'=> "<<lg<<"/"<<z<<"..."<<rg<<"/"<<z<<endl;
for(j=1;j<size;j++)
 { lg2=addp(lg2,"1"); W[p[j-1]]++;
 for(k=0;k<W.size();k++) if(W[k]>0) k<32?cout<<"\'\\"<<int(k):cout<<"\'"<<char(k),cout<<"\': "<<W[k]<<" ";
 for(s=i=0;r[i]!=p[j];s+=W[r[i++]]);
 lg=mulp(lg,lg2); rg=mulp(rg,lg2); z=mulp(z,lg2); { stringstream out; out<<s; s1=out.str(); }
 s+=W[r[i]]; { stringstream out; out<<s; s2=out.str(); }
 tlg=addp(lg,mulp(s1,div(subp(rg,lg),lg2,ch))); trg=addp(lg,mulp(s2,div(subp(rg,lg),lg2,ch)));
 lg=tlg; rg=trg; (p[j]<32?cout<<"\'\\"<<int(p[j]):cout<<"\'"<<p[j]),cout<<"\'=>
"<<lg<<"/"<<z<<"..."<<rg<<"/"<<z<<endl;
 }
for(zn=z,ch="0",zz="2";;zz=mulp(zz,"2"))
 { tlg=mulp(lg,zz);trg=mulp(rg,zz);tzn=mulp(zn,zz);
 ch=divp(tlg,zn,t); tch=mulp(ch,zn);
 cout<<lg<<"/"<<zn<<" < "<<ch<<"/"<<zz<<" < "<<rg<<"/"<<zn<<" - test"<<endl;
 cout<<tlg<<"/"<<tzn<<" < "<<tch<<"/"<<tzn<<" < "<<trg<<"/"<<tzn<<" - общий знаменатель"<<endl;
 for(;ltp(tch,trg);ch=addp(ch,"1"),tch=mulp(ch,zn)  );
 ch=subp(ch,"1"),tch=mulp(ch,zn);
 cout<<tlg<<"/"<<tzn<<" < "<<tch<<"/"<<tzn<<" < "<<trg<<"/"<<tzn<<" - итог test"<<endl;
 cout<<lg<<"/"<<zn<<" < "<<ch<<"/"<<zz<<" < "<<rg<<"/"<<zn<<" - итог test"<<endl;
 if(!ltp(tch,tlg)&&!eq(tch,tlg)) break;
 }
cout<<"ch/zz="<<ch<<"/"<<zz<<endl; for(t="",lg2="";!eq(zz,"1");) zz=divp(zz,"2",t),ch=divp(ch,"2",t),lg2=t+lg2;
cout<<"---------------------------------------=>"<<lg2<<endl; delete [] p; delete [] r;return 0;
}
```

Подстановочные или словарно-ориентированные алгоритмы сжатия информации. Методы Лемпела-Зива

Методы Шеннона-Фано, Хаффмана и арифметического кодирования называют статистическими. Далее рассматриваются словарные алгоритмы, которые носят менее математически обоснованный, но более практичный характер.

Алгоритм LZ77

Алгоритм LZ77 был опубликован в 1977 г. Разработан израильскими математиками Якобом Зивом и Авраамом Лемпелом.

Основная идея LZ77 состоит в том, что второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на её первое вхождение.

Алгоритм LZ77 использует уже просмотренную часть сообщения как словарь. Чтобы добиться сжатия, он пытается заменить очередной фрагмент сообщения на указатель в содержимое словаря.

Алгоритм LZ77 использует «скользящее» по сообщению окно, разделённое на две неравные части. Первая, большая по размеру, включает уже просмотренную часть сообщения. Вторая, намного меньшая, является буфером, содержащим ещё незакодированные символы входного потока.

Обычно размер окна составляет несколько килобайт, а размер буфера — не более 100 байт. Алгоритм пытается найти в словаре (большей части окна) фрагмент, совпадающий с содержимым буфера.

Алгоритм LZ77 выдаёт коды, состоящие из трёх элементов:

– смещение в словаре относительно его начала подстроки, совпадающей с началом содержимого буфера;

– длина этой подстроки;

– первый символ буфера, следующий за подстрокой.

Пример. Размер окна — 20 символов, словаря — 12 символов, а буфера — 8. Кодируется сообщение «ПРОГРАММНЫЕ ПРОДУКТЫ ФИРМЫ MICROSOFT». Пусть словарь уже заполнен. Тогда он содержит строку «ПРОГРАММНЫЕ », а буфер — строку «ПРОДУКТЫ». Просматривая словарь, алгоритм обнаружит, что совпадающей подстрокой будет «ПРО», в словаре она расположена со смещением 0 и имеет длину 3 символа, а следующим символом в буфере является «Д». Таким образом, выходным кодом будет тройка $\langle 0,3,'Д'\rangle$. После этого алгоритм сдвигает влево всё содержимое окна на длину совпадающей подстроки +1 и одновременно считывает столько же символов из входного потока в буфер. Получаем в словаре строку «РАММНЫЕ ПРОД», в буфере — «УКТЫ ФИР». В данной ситуации совпадающей подстроки обнаружить не удаться и алгоритм выдаст код $\langle 0,0,'У'\rangle$, после чего сдвинет окно влево на один символ. Затем словарь будет содержать «АММНЫЕ ПРОДУ», а буфер «КТЫ ФИРМ». И т. д. Декодирование кодов LZ77 проще их получения, так как не нужно осуществлять поиск в словаре. Недостатки LZ77:

1) с ростом размеров словаря скорость работы алгоритма кодера пропорционально замедляется;

2) кодирование одиночных символов очень неэффективно.

Пример. Закодировать по алгоритму LZ77 строку «КРАСНАЯ КРАСКА».

СЛОВАРЬ(8)	БУФЕР(5)	КОД
«........»	«КРАСН»	<0,0‘К’>
«.......К»	«РАСНА»	<0,0‘Р’>
«......КР»	«АСНАЯ»	<0,0‘А’>
«.....КРА»	«СНАЯ »	<0,0‘С’>
«....КРАС»	«НАЯ К»	<0,0‘Н’>
«...КРАСН»	«АЯ КР»	<5,1‘Я’>
«.КРАСНАЯ»	« КРАС»	<0,0‘ ’>
«КРАСНАЯ »	«КРАСК»	<0,4‘К’>
«АЯ КРАСК»	«А....»	<0,0‘А’>

Рис. 3.10

В последней строчке, буква «А» берётся не из словаря, так как она последняя.

Длина кода вычисляется следующим образом: длина подстроки не может быть больше размера буфера, а смещение не может быть больше размера словаря минус 1. Следовательно, длина двоичного кода смещения будет округлённым в большую сторону $\log_2(\text{размер словаря})$, а длина двоичного кода для длины подстроки будет округлённым в большую сторону $\log_2(\text{размер буфера}+1)$. А символ кодируется 8 битами (например ASCII+) В последнем примере длина полученного кода равна $9\cdot(3+3+8)=126$ бит, против 112 бит исходной длины строки. Подробный разбор на рис. 3.11.

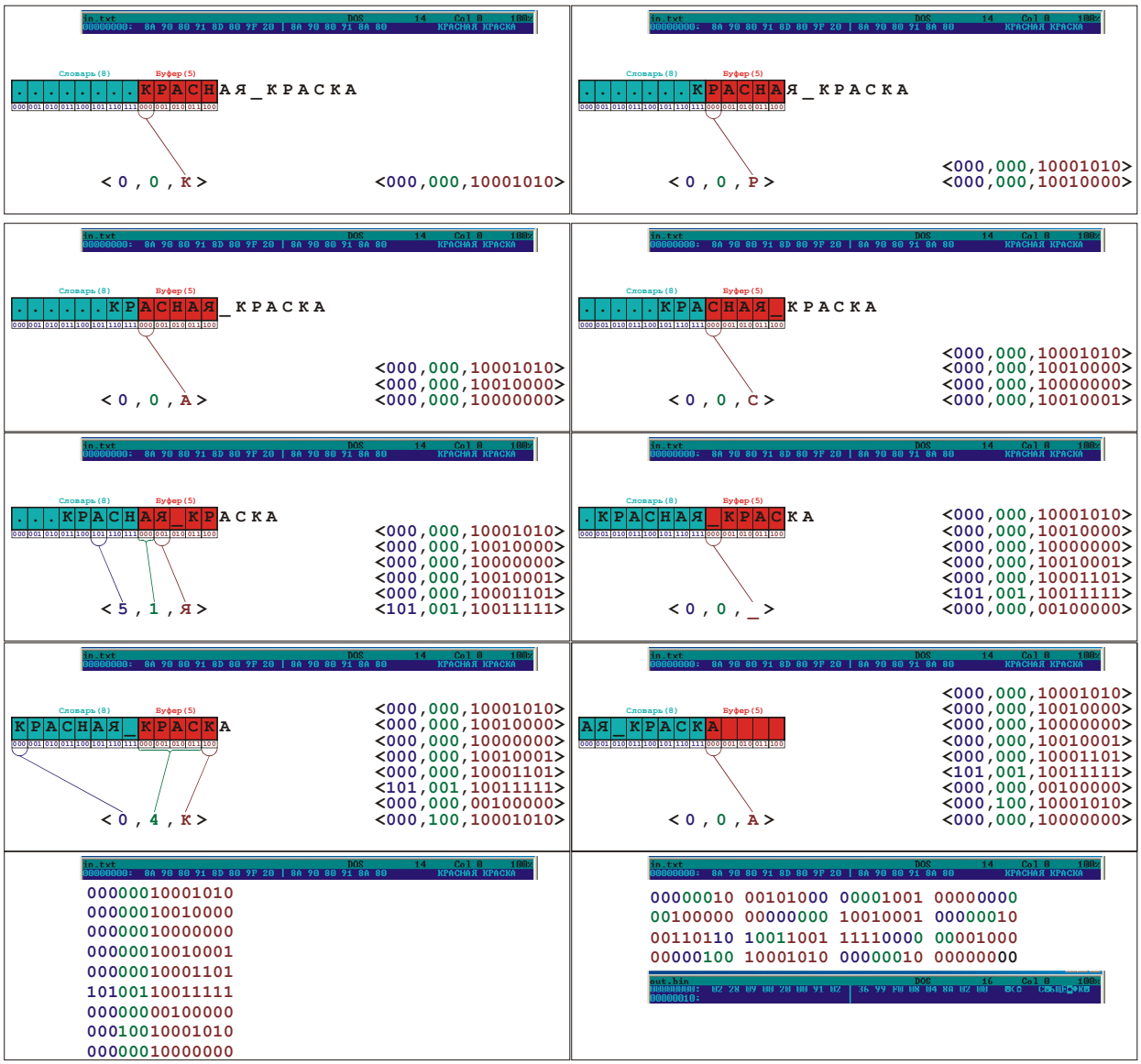


Рис. 3.11

Алгоритм LZSS

В 1982 году Сторером (Storer) и Шиманским (Szimanski) на базе LZ77 был разработан алгоритм LZSS.

Код, выдаваемый алгоритмом LZSS, начинается с однобитового префикса, различающего собственно код от незакодированного символа. Код состоит из пары: смещение и длина, такими же как и для LZ77. В LZSS окно сдвигается ровно на длину найденной подстроки или на 1, если не найдено вхождение подстроки из буфера в словарь. Длина подстроки в LZSS всегда больше нуля, поэтому длина двоичного кода для длины подстроки — это округлённый до большего целого двоичный логарифм от длины буфера.

Пример. Закодировать по алгоритму LZSS строку «КРАСНАЯ КРАСКА».

СЛОВАРЬ(8)	БУФЕР(5)	КОД	ДЛИНА КОДА
«........»	«КРАСН»	0‘К’	9
«.......К»	«РАСНА»	0‘Р’	9
«......КР»	«АСНАЯ»	0‘А’	9
«.....КРА»	«СНАЯ »	0‘С’	9
«....КРАС»	«НАЯ К»	0‘Н’	9
«...КРАСН»	«АЯ КР»	1<5,1>	7
«..КРАСНА»	«Я КРА»	0‘Я’	9
«.КРАСНАЯ»	« КРАС»	0‘ ’	9
«КРАСНАЯ »	«КРАСК»	1<0,4>	7
«НАЯ КРАС»	«КА...»	1<4,1>	7
«АЯ КРАСК»	«А....»	1<0,1>	7

Рис.3.12

Здесь длина полученного кода равна $7 \cdot 9 + 4 \cdot 7 = 91$ бит.

Алгоритмы LS77 и LZSS обладают следующими очевидными недостатками:

1) невозможность кодирования подстрок, отстоящих друг от друга на расстоянии, большем длины словаря;

2) длина подстроки, которую можно закодировать, ограничена размером буфера.

Если механически чрезмерно увеличивать размеры словаря и буфера, то это приведёт к снижению эффективности кодирования, так как с ростом этих величин будут расти и длины кодов для смещения и длины, что сделает коды для коротких подстрок недопустимо большими. Кроме того, резко увеличится время работы алгоритма-кодера.

В 1978 году авторами LZ77 был разработан алгоритм LZ78, лишённый названных недостатков. Алгоритм LZ78 не использует «скользящее» окно, он хранит словарь из уже просмотренных фраз. При старте алгоритма этот словарь содержит только одну пустую строку (строку длины нуль). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введённого символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введённая подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Ключевым для размера получаемых кодов является размер словаря во фразах, потому что каждый код при кодировании по методу LZ78 содержит номер фразы в словаре. Из последнего следует, что эти коды имеют постоянную длину, равную округлённому в большую сторону двоичному логарифму размера словаря+8 (это количество бит в байт-коде расширенного ASCII).

На рис. 3.13 показан подробный процесс кодирования фразы «КРАСНАЯ КРАСКА». Эти пояснения помогут при отладке данного алгоритма.

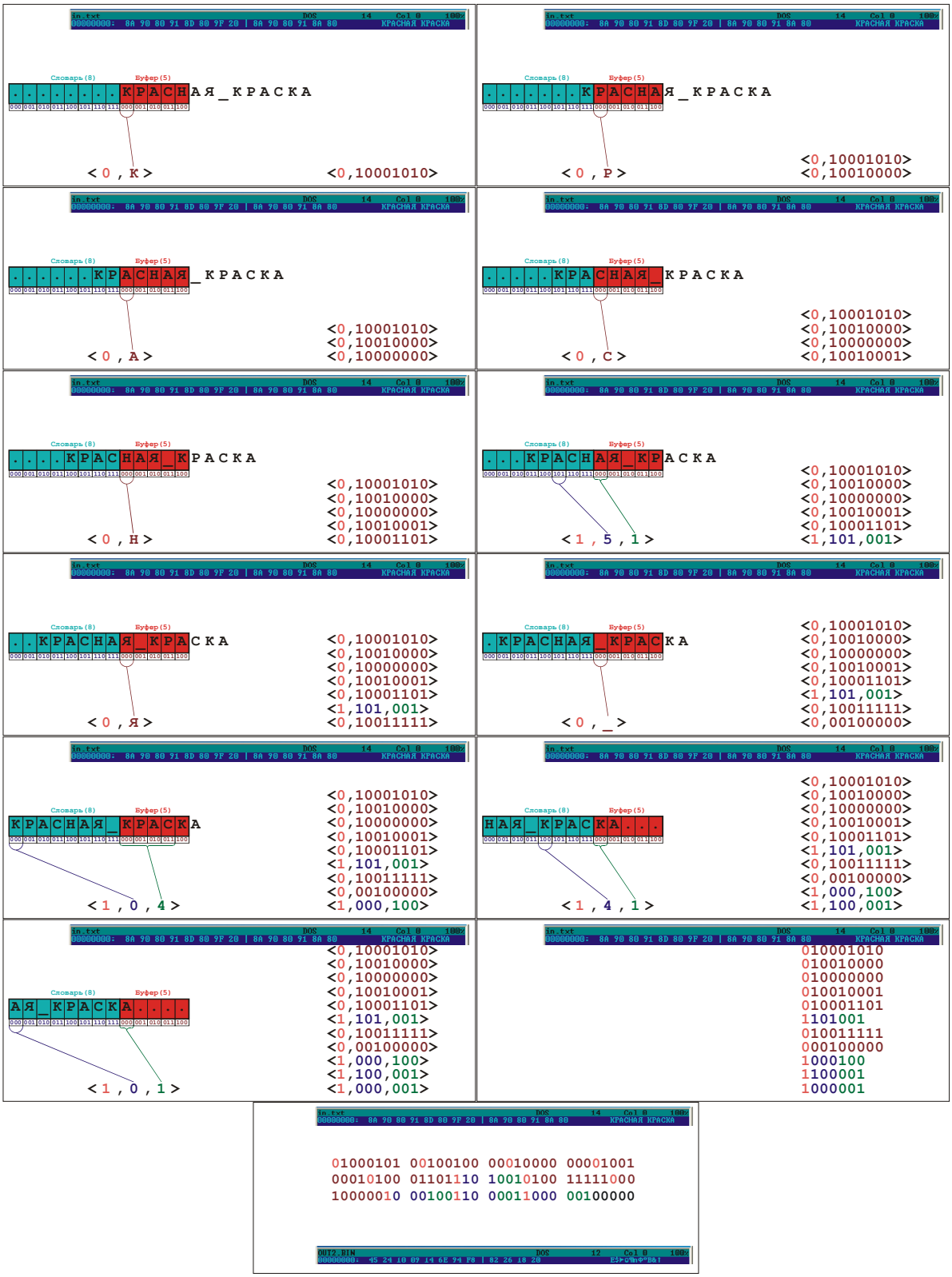

Словарь (8)
Буфер (5)
. . . . . КРАСНАЯ_КРАСКА
< 0 , К >
<0,10001010>
. . . . . КРАСНАЯ_КРАСКА
< 0 , Р >
<0,10001010>
<0,10010000>
. . . . . КРАСНАЯ_КРАСКА
< 0 , А >
<0,10001010>
<0,10010000>
<0,10000000>
. . . . . КРАСНАЯ_КРАСКА
< 0 , С >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
. . . . КРАСНАЯ_КРАСКА
< 0 , Н >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
<0,10001101>
. . . КРАСНАЯ_КРАСКА
< 1 , 5 , 1 >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
<0,10001101>
<1,101,001>
. . КРАСНАЯ_КРАСКА
< 0 , Я >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
<0,10001101>
<1,101,001>
<0,10011111>
. КРАСНАЯ_КРАСКА
< 0 , _ >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
<0,10001101>
<1,101,001>
<0,10011111>
<0,00100000>
КРАСНАЯ_КРАСКА
< 1 , 0 , 4 >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
<0,10001101>
<1,101,001>
<0,10011111>
<0,00100000>
<1,000,100>
НАЯ_КРАСКА. . .
< 1 , 4 , 1 >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
<0,10001101>
<1,101,001>
<0,10011111>
<0,00100000>
<1,000,100>
<1,100,001>
АЯ_КРАСКА. . . .
< 1 , 0 , 1 >
<0,10001010>
<0,10010000>
<0,10000000>
<0,10010001>
<0,10001101>
<1,101,001>
<0,10011111>
<0,00100000>
<1,000,100>
<1,100,001>
<1,000,001>
010001010
010010000
010000000
010010001
010001101
1101001
010011111
000100000
1000100
1100001
1000001
01000101 00100100 00010000 00001001
00010100 01101110 10010100 11111000
10000010 00100110 00011000 00100000

Рис.3.13

Алгоритм LZ78

Пример. Закодировать по алгоритму LZ78 строку «КРАСНАЯ КРАСКА», используя словарь длиной 16 фраз. Указатель на любую фразу такого словаря — это число от 0 до 15, для его кодирования достаточно четырёх бит.

ВХОДНАЯ ФРАЗА (В СЛОВАРЬ)	КОД	ПОЗИЦИЯ СЛОВАРЯ
«»		0
«К»	<0‘К’>	1
«Р»	<0‘Р’>	2
«А»	<0‘А’>	3
«С»	<0‘С’>	4
«Н»	<0‘Н’>	5
«АЯ»	<3‘Я’>	6
« »	<0‘ ’>	7
«КР»	<1‘Р’>	8
«АС»	<3‘С’>	9
«КА»	<1‘А’>	10

Рис. 3.14

В последнем примере длина полученного кода равна $10 \cdot (4+8) = 120$ битам.

На рис. 3.15 показан подробный процесс кодирования фразы «КРАСНАЯ КРАСКА». Эти пояснения помогут при отладке данного алгоритма.

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1			
0010	2			
0011	3			
0100	4			
0101	5			
0110	6			
0111	7			
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2			
0011	3			
0100	4			
0101	5			
0110	6			
0111	7			
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3			
0100	4			
0101	5			
0110	6			
0111	7			
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4			
0101	5			
0110	6			
0111	7			
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4	«С»	<0,С>	<0000,10010001>
0101	5			
0110	6			
0111	7			
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4	«С»	<0,С>	<0000,10010001>
0101	5	«Н»	<0,Н>	<0000,10001101>
0110	6			
0111	7			
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4	«С»	<0,С>	<0000,10010001>
0101	5	«Н»	<0,Н>	<0000,10001101>
0110	6	«АЯ»	<3,Я>	<0011,10011111>
0111	7			
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4	«С»	<0,С>	<0000,10010001>
0101	5	«Н»	<0,Н>	<0000,10001101>
0110	6	«АЯ»	<3,Я>	<0011,10011111>
0111	7	«_»	<0,_>	<0000,00100000>
1000	8			
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4	«С»	<0,С>	<0000,10010001>
0101	5	«Н»	<0,Н>	<0000,10001101>
0110	6	«АЯ»	<3,Я>	<0011,10011111>
0111	7	«_»	<0,_>	<0000,00100000>
1000	8	«КР»	<1,Р>	<0001,10010000>
1001	9			
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4	«С»	<0,С>	<0000,10010001>
0101	5	«Н»	<0,Н>	<0000,10001101>
0110	6	«АЯ»	<3,Я>	<0011,10011111>
0111	7	«_»	<0,_>	<0000,00100000>
1000	8	«КР»	<1,Р>	<0001,10010000>
1001	9	«АС»	<3,С>	<0011,10010001>
1010	10			
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
0000	0	«»	—	—
0001	1	«К»	<0,К>	<0000,10001010>
0010	2	«Р»	<0,Р>	<0000,10010000>
0011	3	«А»	<0,А>	<0000,10000000>
0100	4	«С»	<0,С>	<0000,10010001>
0101	5	«Н»	<0,Н>	<0000,10001101>
0110	6	«АЯ»	<3,Я>	<0011,10011111>
0111	7	«_»	<0,_>	<0000,00100000>
1000	8	«КР»	<1,Р>	<0001,10010000>
1001	9	«АС»	<3,С>	<0011,10010001>
1010	10	«КА»	<1,А>	<0001,10000000>
1011	11			
1100	12			
1101	13			
1110	14			
1111	15			

```
00001000 10100000 10010000 00001000
00000000 10010001 00001000 11010011
10011111 00000010 00000001 10010000
00111001 00010001 10000000
```

```
00001000 10100000 10010000 00001000
00000000 10010001 00001000 11010011
10011111 00000010 00000001 10010000
00111001 00010001 10000000
```

Рис 3.15

Алгоритм Лемпела-Зива для двоичных символов

В технике экономного кодирования более типичной является ситуация, когда о некотором избыточном источнике известен лишь его алфавит, но не известна его статистика (распределение вероятностей последовательностей символов). Так, например, может стоять задача разработки некоторого универсального сжимающего двоичного префиксного кода для передачи текста на различных языках, заданных каждый своим алфавитом, или для сжатия данных различного рода.

Основная идея этого алгоритма заключается в том, что последовательность символов источника разбивается на максимально короткие различимые цепочки, которые не встречались раньше, а затем эти цепочки кодируются равномерным кодом. Например, если источник выдаёт двоичную последовательность из 18 символов: 101100110001111000, то она будет разбита на такие цепочки: 1, 0, 11, 00, 110, 001, 111, 000. При таком разбиении все префиксы конкретной цепочки могут находиться лишь слева, и две цепочки с одинаковым префиксом всегда отличаются в последнем символе (бите). В каждой цепочке кодируется её префикс равномерным двоичным кодом и один бит используется для кодирования последнего (справа) символа цепочки.

Если $C(n)$ означает число различных цепочек разбиения для данной последовательности из n символов (в нашем примере $C(n)=8$), то требуется $\log_2 C(n)$ бит, чтобы закодировать номер префикса к данной цепочке (в нашем примере 3 бита) и ещё один бит для описания последнего элемента этой цепочки. Для приведённой выше цепочки и её разбиения получаем следующий равномерный код: $(000,1)$, $(000,0)$, $(001,1)$, $(010,0)$, $(011,0)$, $(100,1)$, $(011,1)$, $(100,0)$. Декодирование такого кода однозначное. В нашем примере кодовые комбинации содержат 32 символа (4×8) вместо исходных 18 символов, то есть получилось растяжение вместо сжатия. Однако для длинных последовательностей сообщений эффективность алгоритма растёт, и при $n\to\infty$ сжатие сообщений приближается к своему пределу, поскольку доказано, что для любого стационарного источника сообщений при использовании двоичного кода Зива-Лемпела среднее число кодовых символов на один символ источника стремится к величине энтропии.

На рис. 3.16 показан подробный процесс кодирования двоичной последовательности «101100110001111000». Эти пояснения помогут при отладке данного алгоритма.

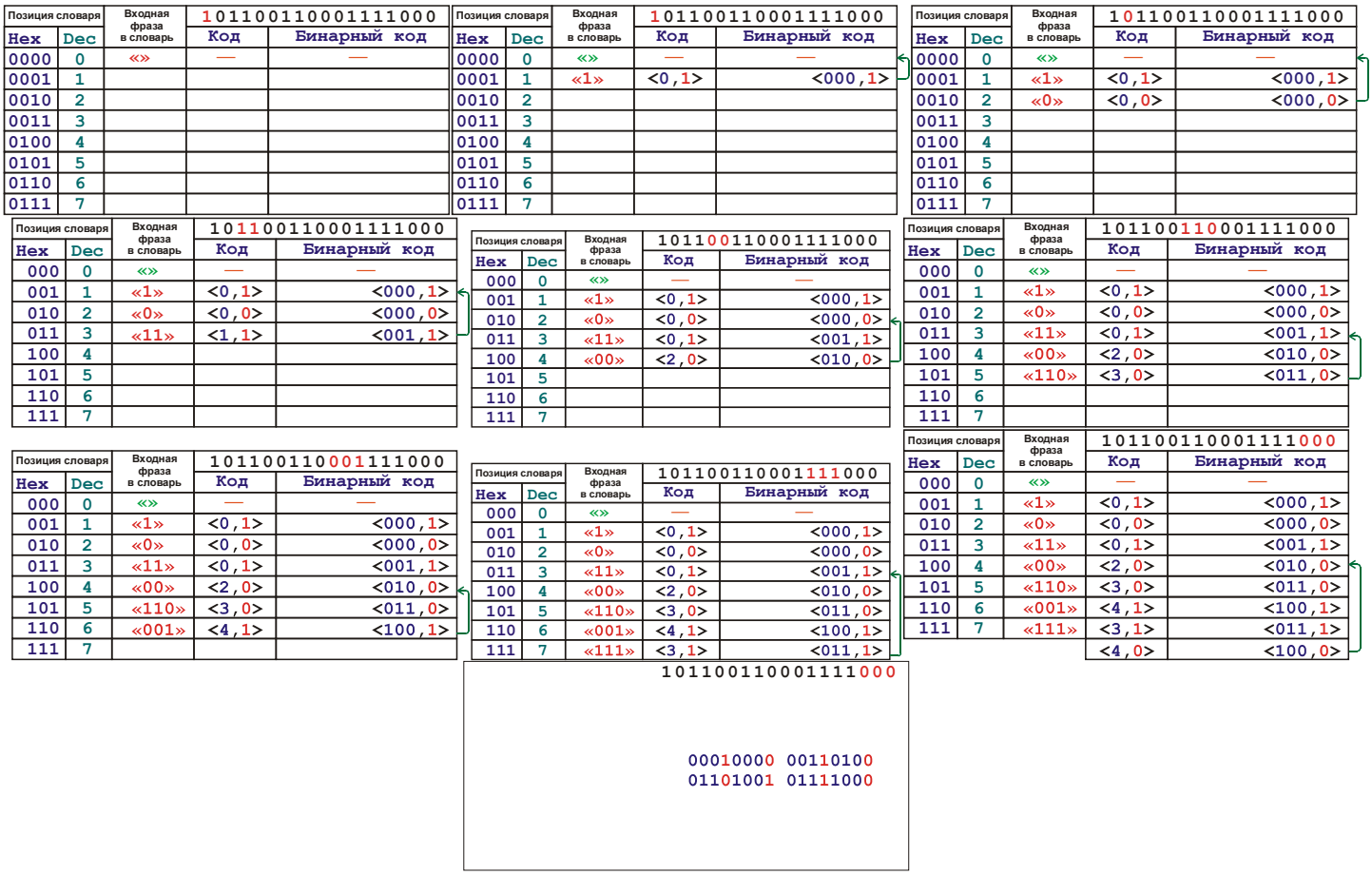


Рис. 3.16

Алгоритм LZW

В 1984 году Уэлчем (Welch) был создан алгоритм LZW путём модификации LZ78.

Пошаговое описание алгоритма-кодера.

Шаг 1. Инициализация словаря всеми возможными односимвольными фразами (обычно 256 символами расширенного ASCII). Инициализация входной фразы w первым символом сообщения.

Шаг 2. Считать очередной символ K из кодируемого сообщения.

Шаг 3. Если КОНЕЦ_СООБЩЕНИЯ

Выдать код для w

Конец

Если фраза wK уже есть в словаре

Присвоить входной фразе значение wK

Перейти к Шагу 2

Иначе

Выдать код w

Добавить wK в словарь

Присвоить входной фразе значение K

Перейти к Шагу 2

Как и в случае с LZ78 для LZW ключевым для размера получаемых кодов является размер словаря во фразах: LZW-коды имеют постоянную длину, равную округлённую в большую сторону двоичному логарифму размера словаря.

Пример. Закодировать строку «КРАСНАЯ КРАСКА». Размер словаря — 500 фраз.

ВХОДНАЯ ФРАЗА, wK (В СЛОВАРЬ)	КОД для w	ПОЗИЦИЯ СЛОВАРЯ
ASCII+		0-255
«КР»	0‘К’	256
«РА»	0‘Р’	257
«АС»	0‘А’	258
«СН»	0‘С’	259
«НА»	0‘Н’	260
«АЯ»	0‘А’	261
«Я »	0‘Я’	262
« К»	0‘ ’	263
«КРА»	<256>	264
«АСК»	<258>	265
«КА»	0‘К’	266
«А»	0‘А’	

Рис. 3.17

В этом примере длина полученного кода равна $12 \cdot 9 = 108$ битам.

При переполнении словаря, то есть когда необходимо внести новую фразу в полностью заполненный словарь, из него удаляют либо наиболее редко используемую фразу, либо все фразы, отличающиеся от одиночного символа. Для LZ78 возможна и полная очистка словаря.

Эффективность кодирования источника (сжатия данных) оценивается через его избыточность следующим соотношением: $\eta_и = 1 - \rho_и$.

Для русского языка $\rho_и \approx 0{,}75$ и, следовательно, максимальная эффективность экономного кодирования $\eta_и \approx 0{,}25$.

На рис. 3.18 показан подробный процесс кодирования фразы «КРАСНАЯ КРАСКА». Эти пояснения помогут при отладке данного алгоритма.

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256			
100000001	257			
100000010	258			
100000011	259			
100000100	260			
100000101	261			
100000110	262			
100000111	263			
100001000	264			
100001001	265			
100001010	266			

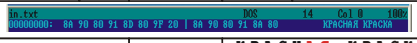
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<110001010>
100000001	257			
100000010	258			
100000011	259			
100000100	260			
100000101	261			
100000110	262			
100000111	263			
100001000	264			
100001001	265			
100001010	266			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<110001010>
100000001	257	«РА»	<0,Р>	<110010000>
100000010	258			
100000011	259			
100000100	260			
100000101	261			
100000110	262			
100000111	263			
100001000	264			
100001001	265			
100001010	266			

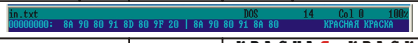
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<110001010>
100000001	257	«РА»	<0,Р>	<110010000>
100000010	258	«АС»	<0,А>	<100000000>
100000011	259			
100000100	260			
100000101	261			
100000110	262			
100000111	263			
100001000	264			
100001001	265			
100001010	266			

Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260			
100000101	261			
100000110	262			
100000111	263			
100001000	264			
100001001	265			
100001010	266			

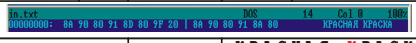
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261			
100000110	262			
100000111	263			
100001000	264			
100001001	265			
100001010	266			



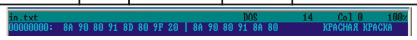
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261	«АЯ»	<0,А>	<010000000>
100000110	262			
100000111	263			
100001000	264			
100001001	265			
100001010	266			



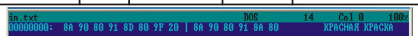
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261	«АЯ»	<0,А>	<010000000>
100000110	262	«Я_»	<0,Я>	<010011111>
100000111	263			
100001000	264			
100001001	265			
100001010	266			



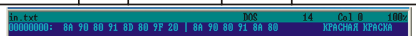
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261	«АЯ»	<0,А>	<010000000>
100000110	262	«Я_»	<0,Я>	<010011111>
100000111	263	«_К»	<0,_>	<000100000>
100001000	264			
100001001	265			
100001010	266			



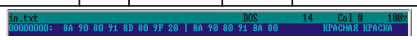
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261	«АЯ»	<0,А>	<010000000>
100000110	262	«Я_»	<0,Я>	<010011111>
100000111	263	«_К»	<0,_>	<000100000>
100001000	264	«КРА»	<256>	<100000000>
100001001	265			
100001010	266			



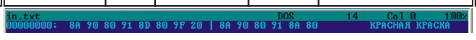
Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261	«АЯ»	<0,А>	<010000000>
100000110	262	«Я_»	<0,Я>	<010011111>
100000111	263	«_К»	<0,_>	<000100000>
100001000	264	«КРА»	<256>	<100000000>
100001001	265	«АСК»	<258>	<100000010>
100001010	266			



Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261	«АЯ»	<0,А>	<010000000>
100000110	262	«Я_»	<0,Я>	<010011111>
100000111	263	«_К»	<0,_>	<000100000>
100001000	264	«КРА»	<256>	<100000000>
100001001	265	«АСК»	<258>	<100000010>
100001010	266	«КА»	<0,К>	<010001010>



Позиция словаря		Входная фраза в словарь	КРАСНАЯ_КРАСКА	
Hex	Dec		Код	Бинарный код
000000000-011111111	0-255	«»	—	—
100000000	256	«КР»	<0,К>	<010001010>
100000001	257	«РА»	<0,Р>	<010010000>
100000010	258	«АС»	<0,А>	<010000000>
100000011	259	«СН»	<0,С>	<010010001>
100000100	260	«НА»	<0,Н>	<010001101>
100000101	261	«АЯ»	<0,А>	<010000000>
100000110	262	«Я_»	<0,Я>	<010011111>
100000111	263	«_К»	<0,_>	<000100000>
100001000	264	«КРА»	<256>	<100000000>
100001001	265	«АСК»	<258>	<100000010>
100001010	266	«КА»	<0,К>	<010001010>
		«А»	<0,А>	<010000000>



```
01000101 00100100 00010000 00001001
00010100 01101010 00000001 00111110
00100000 10000000 01000000 10010001
01001000 00000000
```

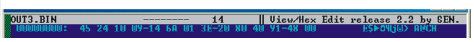


Рис. 3.18

Лабораторная работа №4. Помехоустойчивое кодирование

Реализовать программным способом задачу помехоустойчивого кодирования. Согласно блочной модели, приведённой на рис 4.1 при наличии готовых реализаций генератора символов и сравнивающего устройства (на рис. 4.1 обозначены как «генератор» и «сравнитель») требуется создать дополнительно три модуля: помехоустойчивый кодер, соответствующий ему декодер и модель дискретного канала.

Рис. 4.1. Блочная модель для проверки кодера и декодера помехоустойчивого кодирования

На рис. 4.1 блок «информация об ОС» представляет собой текстовый или бинарный файл с информацией об обратной связи (ОС). Обратная связь требуется при повторной перечаче кодовых блочных последовательностей в режиме обнаружения.

Так как в заданиях будут использоваться коды с разной длиной кодового слова и информационной части в нём, то предлагается векторно-матричный подход к моделированию блочных кодов.

Векторно-матричное представление кодов

Символы исходного сообщения и кодового слова можно рассматривать как элементы соответствующих векторов. Обозначим информационный вектор как

$$\mathbf{a} = (a_0\ a_1\ ...\ a_{k-1})$$

а кодовый вектор как

$$\mathbf{b} = (b_0\ b_1\ ...\ b_{n-1})$$

Для[7] линейных блоковых кодов операцию кодирования можно представить совокупностью линейных уравнений вида

$$b_i = a_{0,i}g_{0,i} + a_{1,i}g_{1,i} + ... + a_{k-1,i}g_{k-1,i},\ \ i = 0, 1, ..., n-1$$

В двоичном случае коэффициенты $g_{i,j}$ принимают значения 0 или 1. Эти линейные уравнения можно переписать в векторно-матричной форме:

$$\mathbf{b} = \mathbf{aG}$$

Матрица $\mathbf{G}$ называется *порождающей матрицей* кода:

$$\mathrm{G} = \begin{Vmatrix} g_0 \\ g_1 \\ ... \\ g_{k-1} \end{Vmatrix} = \begin{Vmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ ... & ... & ... & ... \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{Vmatrix}.$$

Таким образом, произвольное слово представляет собой линейную комбинацию векторов g_i — строк матрицы G:

$$\mathbf{b} = a_0\mathrm{g}_0 + a_1\mathrm{g}_1 + ... + a_{k-1}\mathrm{g}_{k-1}.$$

Векторы g_i должны выбираться таким образом, чтобы соблюдалось условие их линейной независимости. Такие векторы называются *базисом* (n,k) кода. Совокупность базисных векторов не единственная и, следовательно, матрица G не уникальна.

Любую порождающую матрицу (n,k) кода путём проведения операций над строками и перестановкой столбцов можно свести к систематической форме:

$$\mathrm{G} = \|\mathrm{I} : \mathrm{P}\| = \left\| \begin{array}{cccc:cccc} 1 & 0 & ... & 0 & p_{0,0} & p_{0,1} & \cdots & p_{0,r-1} \\ 0 & 1 & ... & 0 & p_{1,0} & p_{1,1} & \cdots & p_{1,r-1} \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 1 & p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,r-1} \end{array} \right\|.$$

Здесь $\mathbf{I}$ — единичная матрица, P — матрица, которая определяет r проверочных символов. Такую форму записи проверочной матрицы называют *канонической*. После умножения информационного вектора на порождающую матрицу в канонической форме, получается кодовый вектор, в котором первые k элементов — информационные, а последние — проверочные.

Любой несистематический линейный блочный (n,k) код, полученный с помощью несистематической порождающей матрицы, эквивалентен (n,k) коду с соответствующей систематической порождающей матрицей.

С любым линейным (n,k) кодом связан *дуальный код* $(n,n-k)$. Дуальный код является линейным кодом с 2^{n-k} кодовыми векторами, которые образуют нуль-пространство по отношению к (n,k) коду. Порождающая матрица $\mathbf{H}$ дуального кода состоит из $(n-k)$ линейно независимых векторов, выбираемых в нуль-пространстве. То есть любое кодовое слово (n,k) кода ортогонально любому кодовому слову дуального $(n,n-k)$ кода. Следовательно, любое кодовое слово (n,k) кода ортогонально любой строке матрицы $\mathbf{H}$:

$\mathbf{b}\mathbf{H}^{\mathrm{T}} = 0$.

Поскольку это равенство справедливо для любого кодового слова, то $\mathbf{G}\mathbf{H}^{\mathrm{T}} = 0$.

Если линейный (n,k) код является систематическим, то из этого равенства следует, что $\mathbf{H} = [-\mathbf{P}^{\mathrm{T}} : \mathbf{I}]$.

Для двоичных кодов знак «–» может быть опущен, так как операции сложения и вычитания по модулю 2 идентичны.

Матрица $\mathbf{H}$ используется при декодировании (n,k) кода в качестве *проверочной*. Если выполняется условие $\mathbf{b}'\mathbf{H}^{\mathrm{T}} = 0$, значит, принятый кодовый вектор является разрешённым и ошибок нет.

Принятый вектор $\mathbf{b}' = \mathbf{b} + \mathbf{e}$, где $\mathbf{b}$ — переданный вектор, $\mathbf{e}$ — вектор ошибки.

Произведение

$\mathbf{b}'\mathbf{H}^{\mathrm{T}} = (\mathbf{b}+\mathbf{e})\mathbf{H}^{\mathrm{T}} = \mathbf{b}\mathbf{H}^{\mathrm{T}} + \mathbf{e}\mathbf{H}^{\mathrm{T}} = \mathbf{e}\mathbf{H}^{\mathrm{T}} = \mathbf{c}$.

Вектор $\mathbf{c}$ называется *вектором синдрома*. Число элементов вектора $\mathbf{c}$ равняется числу проверочных символов. Отличие синдрома от нуля всегда указывает на наличие ошибки. Кроме того, в режиме исправления по виду синдрома можно определить вектор ошибки и, следовательно, исправить ошибки в принятой кодовой комбинации. При декодировании все возможные варианты синдрома сводятся в *таблицу синдромов*. Пример таблицы синдромов:

Синдром	Номер ошибочного разряда
001	4
010	5
011	3
…	…

Рис. 4.2.

Синдром $\mathbf{c}$ является характеристикой вектора ошибок, а не переданного кодового вектора. Хотя в принципе возможно 2^n вариантов ошибок, из них лишь $2^{n-k} = 2^r$ будут синдромными. Следовательно, разные ошибки могут приводить к одинаковым синдромам.

Моделирование линейных блочных систематических кодов с использованием векторно-матричного представления

Пусть задан линейный код (8,4): `n=8 k=4 r=4`

`Порождающая матрица:` $\mathbf{G} = \begin{Vmatrix} 10000111 \\ 01001011 \\ 00101101 \\ 00011110 \end{Vmatrix}$

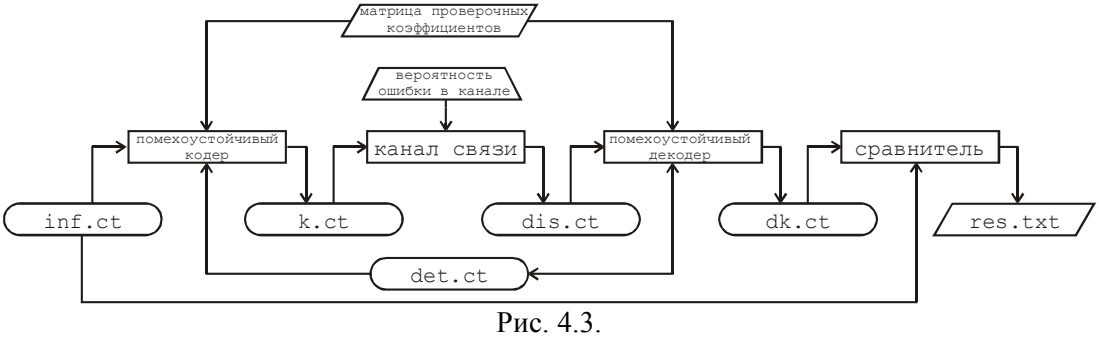


Рис. 4.3.

Для этого кода информационные и кодовые комбинации показаны на рис. 4.4

```
0000: 00000000
0001: 00011110
0010: 00101101
0011: 00110011
0100: 01001011
0101: 01010101
0110: 01100110
0111: 01111000
1000: 10000111
1001: 10011001
1010: 10101010
1011: 10110100
1100: 11001100
1101: 11010010
1110: 11100001
1111: 11111111
```

Рис. 4.4

Минимальное расстояние по Хеммингу для этого кода $d_{\min} = 4$. Это означает, что данный код в режиме обнаружения гарантированно обнаруживает до 3 ошибок в кодовой комбинации, а в режиме исправления исправляет не более 1 ошибки. Однако такой код не является совершенным, и некоторые неразрешённые комбинации в режиме исправления будут сигнализировать о необходимости повторной передачи блока.

Проверочная транспонированная матрица: $\mathbf{H}^{\mathrm{T}} = \begin{Vmatrix} 0111 \\ 1011 \\ 1101 \\ 1110 \\ 1000 \\ 0100 \\ 0010 \\ 0001 \end{Vmatrix}$

Вектор-синдром в десятичной и двоичной системе счисления, а также маска для бита, который является ошибочным

```
        76543210 номера разрядов
7 0111 10000000
111011 01000000
131101 00100000
141110 00010000
8 1000 00001000
4 0100 00000100
2 0010 00000010
1 0001 00000001
```

Рис.4.5

Все остальные варианты синдромов соответствуют обнаружению ошибки

Рассмотрим подробно процесс кодирования двух информационных блоков и ситуации, когда кодовые комбинации принимаются без ошибок, а также при наличии одной ошибки в некотором разряде кодовой комбинации.

Кодирование байта информации: ср1251 «Ы»: 11011011 он разбивается на два полубайта: 1101 1011

Полубайты становятся двумя последовательными 4-разрядными информационными комбинациями. Умножаем 4-разрядные информационные комбинации на порождающую матрицу, получаем 8-разрядные разрешённые кодовые комбинации:

$$\mathbf{a}_{13}\mathbf{G}=\|1101\|\begin{Vmatrix}10000111\\01001011\\00101101\\00011110\end{Vmatrix}=\begin{Vmatrix}1\cdot1\oplus1\cdot0\oplus0\cdot0\oplus1\cdot0=1\oplus0\oplus0\oplus0=1\\1\cdot0\oplus1\cdot1\oplus0\cdot0\oplus1\cdot0=0\oplus1\oplus0\oplus0=1\\1\cdot0\oplus1\cdot0\oplus0\cdot1\oplus1\cdot0=0\oplus0\oplus0\oplus0=0\\1\cdot0\oplus1\cdot0\oplus0\cdot0\oplus1\cdot1=0\oplus0\oplus0\oplus1=1\\1\cdot0\oplus1\cdot1\oplus0\cdot1\oplus1\cdot1=0\oplus1\oplus0\oplus1=0\\1\cdot1\oplus1\cdot0\oplus0\cdot1\oplus1\cdot1=1\oplus0\oplus0\oplus1=0\\1\cdot1\oplus1\cdot1\oplus0\cdot0\oplus1\cdot1=1\oplus1\oplus0\oplus1=1\\1\cdot1\oplus1\cdot1\oplus0\cdot1\oplus1\cdot0=1\oplus1\oplus0\oplus0=0\end{Vmatrix}=\|11010010\|=\mathbf{b}_{13}$$

$$\mathbf{a}_{11}\mathbf{G}=\|1011\|\begin{Vmatrix}10000111\\01001011\\00101101\\00011110\end{Vmatrix}=\begin{Vmatrix}1\cdot1\oplus0\cdot0\oplus1\cdot0\oplus1\cdot0=1\oplus0\oplus0\oplus0=1\\1\cdot0\oplus0\cdot1\oplus1\cdot0\oplus1\cdot0=0\oplus0\oplus0\oplus0=0\\1\cdot0\oplus0\cdot0\oplus1\cdot1\oplus1\cdot0=0\oplus0\oplus1\oplus0=1\\1\cdot0\oplus0\cdot0\oplus1\cdot0\oplus1\cdot1=0\oplus0\oplus0\oplus1=1\\1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus1\cdot1=0\oplus0\oplus1\oplus1=0\\1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus1\cdot1=1\oplus0\oplus1\oplus1=1\\1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus1\cdot1=1\oplus0\oplus0\oplus1=0\\1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus1\cdot0=1\oplus0\oplus1\oplus0=0\end{Vmatrix}=\|10110100\|=\mathbf{b}_{11}$$

Если в канале нет ошибок, то кодовые комбинации остаются неизменными, то есть разрешёнными. Результат умножения разрешённых кодовых комбинаций на проверочную матрицу даёт нулевой 4-х разрядный синдром

$$\mathbf{b}_{13}\mathbf{H}^{\mathrm{T}}=\|11010010\|\begin{Vmatrix}0111\\1011\\1101\\1110\\1000\\0100\\0010\\0001\end{Vmatrix}=\begin{Vmatrix}1\cdot0\oplus1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot0\oplus0\cdot0=0\oplus1\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus0\cdot0\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0=1\oplus0\oplus1\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0\oplus0\cdot0\oplus1\cdot0\oplus0\cdot1=1\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0\oplus0=0\end{Vmatrix}=\|0000\|=\mathbf{c}_0$$

$$\mathbf{b}_{11}\mathbf{H}^{\mathrm{T}}=\|10110100\|\begin{Vmatrix}0111\\1011\\1101\\1110\\1000\\0100\\0010\\0001\end{Vmatrix}=\begin{Vmatrix}1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0\oplus0\cdot0=0\oplus0\oplus1\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0=1\oplus0\oplus1\oplus1\oplus0\oplus1\oplus0\oplus0=0\\1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus1\cdot0\oplus0\cdot0\oplus1\cdot0\oplus0\cdot1\oplus0\cdot0=1\oplus0\oplus1\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus1\cdot0\oplus0\cdot0\oplus0\cdot0\oplus0\cdot0\oplus1\cdot0=1\oplus0\oplus1\oplus1\oplus0\oplus0\oplus0\oplus0=0\end{Vmatrix}=\|0000\|=\mathbf{c}_0$$

Пусть в канале есть ошибки. Сделаем одну ошибку в 6-м разряде в кодовой комбинации $\mathbf{b}_{13}$ и две ошибки в 4-м и 1-м разряде в кодовой комбинации $\mathbf{b}_{11}$.

$$\mathbf{b}_{13}=\|11010010\|\rightarrow\mathbf{b}'_{13}=\|10010010\|\qquad\mathbf{b}_{11}=\|10110100\|\rightarrow\mathbf{b}''_{11}=\|10100110\|$$

Результат умножения полученных кодовых комбинаций на проверочную матрицу даёт ненулевые 4-х разрядные синдромы

$$\mathbf{b}_{13}\mathbf{H}^{\mathrm{T}}=\|10010010\|\begin{Vmatrix}0111\\1011\\1101\\1110\\1000\\0100\\0010\\0001\end{Vmatrix}=\begin{Vmatrix}1\cdot0\oplus0\cdot1\oplus0\cdot1\oplus1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot0\oplus0\cdot0=0\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=1\\1\cdot1\oplus0\cdot0\oplus0\cdot1\oplus1\cdot1\oplus0\cdot0\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0=0\\1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0\oplus1\cdot1\oplus0\cdot0=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus1\oplus0=1\\1\cdot1\oplus0\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot0\oplus0\cdot0\oplus1\cdot0\oplus0\cdot1=1\oplus0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus1=1\end{Vmatrix}=\|1011\|=\mathbf{c}_{11}$$

Синдром, содержащий комбинацию 1011, есть в таблице синдромов и как раз соответствует

предположению об ошибке в 6-м разряде. Поэтому следующим действием будет инвертирование 6-го разряда в кодовой комбинации (исправление ошибки)

$$\mathbf{b}'_{13}=\left\|1\bar{0}010010\right\|\rightarrow \mathbf{b}_{13}=\left\|\underline{11}010010\right\|\rightarrow 1101$$

после этого старшие 4 разряда являются искомыми, которые и передаются на выход декодера: 1101

$$\mathbf{b}_{11}\mathbf{H}^{\mathrm{T}}=\left\|10100110\right\|\begin{Vmatrix}0111\\1011\\1101\\1110\\1000\\0100\\0010\\0001\end{Vmatrix}=\begin{Vmatrix}1\cdot0\oplus0\cdot1\oplus1\cdot1\oplus0\cdot1\oplus0\cdot1\oplus1\cdot0\oplus1\cdot0\oplus0\cdot0=0\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0=1\\1\cdot1\oplus0\cdot0\oplus1\cdot1\oplus0\cdot1\oplus0\cdot0\oplus1\cdot1\oplus1\cdot0\oplus0\cdot0=1\oplus0\oplus1\oplus0\oplus0\oplus1\oplus0\oplus0=1\\1\cdot1\oplus0\cdot1\oplus1\cdot0\oplus0\cdot1\oplus0\cdot0\oplus1\cdot0\oplus1\cdot1\oplus0\cdot0=1\oplus0\oplus0\oplus0\oplus0\oplus0\oplus1\oplus0=0\\1\cdot1\oplus0\cdot1\oplus1\cdot1\oplus0\cdot0\oplus0\cdot0\oplus1\cdot0\oplus1\cdot0\oplus0\cdot1=1\oplus0\oplus1\oplus0\oplus0\oplus0\oplus0\oplus0=0\end{Vmatrix}=\left\|1100\right\|=\mathbf{c}_{12}$$

Здесь синдром содержит комбинацию 1100, которой нет в таблице синдромов.

Если отсутствует канал обратной связи, то информационные символы считаются стёртыми. Можно их закодировать символами «2»:

$\mathbf{b}''_{11}=\left\|10100110\right\|\rightarrow 2222$. Таким образом, на выходе декодера получается следующая последовательность символов: 11012222

Можно запоминать номера стёртых информационных комбинаций (сформировать из них файл). Тогда при повторной работе кодера на его выходе будут формироваться кодовые комбинации, соответствующие информационным комбинациям, только для тех, порядковые номера которых, записаны в файле.

Последовательность на выходе кодера также может быть подвержена искажениям в виде изменения некоторых бит (с 0 на 1 или с 1 на 0). Декодер также сделает произведение на проверочную матрицу и в случае положительного исхода согласно порядковым номерам в соответствующем файле заменит стёртые символы на искомые. Если останутся стёртые символы, то файл с порядковыми номерами для стёртых информационных комбинаций будет обновлён. При этом размер этого файла либо не изменится (что крайне маловероятно) или станет меньше.

Процесс будет повторяться до тех пор, пока размер файла с порядковыми номерами для стёртых информационных комбинаций не станет равным нулю.

Пример программ на C++, реализующих работу кодера и декодера приведён ниже. Входной файл с матрицей проверочных символов представляет собой текстовый файл.

Листинг помехоустойчивого кодера

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main(int mn,char* nm[])
{ int size,i,j,k,n,r,l,dmin,N; char h,c=0; long long s; long u;
if(mn==5||mn==4); else { cerr<<nm[0]<<" in.ct out.ct p.txt r.lb\n"; return 1; }
ifstream det,pk(nm[3],ios::binary);
if(!pk) { cerr<<"input file \""<<nm[3]<<"\" not open!\n"; return 1; }
ofstream ou(nm[2],ios::binary);
if(!ou) { cerr<<"output file \""<<nm[2]<<"\" not open!\n"; return 1; }
ifstream in(nm[1],ios::binary);
if(!in) { cerr<<"input file \""<<nm[1]<<"\" not open!\n"; return 1; }
h=0x30*((nm[2][strlen(nm[2])-1]|0x20)=='t');
in.seekg(0,ios::end);N=in.tellg();in.seekg(0,ios::beg);
pk.seekg(0,ios::end);size=pk.tellg();pk.seekg(0,ios::beg);
char* p=new char[size+2]; p[size]=0x0A;p[size+1]=0; pk.read(p,size); pk.close();
for(j=0,i=-1;p[++i];) if((p[i]|1)==0x31||p[i]==0x0A) p[j++]=p[i]; p[j]=0;
for(j=0,i=-1;p[++i];c=p[i]) if(p[i]==0x0A&&c==0x0A); else p[j++]=p[i]; p[j]=0;
for(r=0;p[r++]!=0x0A;); k=j/r--; n=k+r;
cout<<"(n,k)=("<<n<<","<<k<<") r="<<r<<endl;
long long* G=new long long[k]; memset(G,0,k*sizeof(long long));
for(l=i=0;i<k;l++,G[i++]>>=1)
{ for(j=0;j<k;G[i]<<=1) G[i]|=(i==j++);
for(j=-1;p[++j]!=0x0A;G[i]<<=1) G[i]|=p[l++]&1;
}
for(i=0;i<(1<<k);i++) for(s=0,l=1<<(k-1),j=0;j<k;s^=G[j++]*!!(i&l),l>>=1);
```

```
for(dmin=n,i=1;i<1<<k;i++)
 { for(l=s=0,j=i;j;j>>=1) s^=G[l++]*!!(j&1);
 for(l=0;s;s>>=1) l+=!!(s&1); if(dmin>l) dmin=l;
 }
if(mn==5) { det.open(nm[4],ios::binary); if(!det) mn=4; }
cout<<"dmin="<<dmin<<endl;
if(mn!=5)
 for(N/=k,i=0;i<N;i++)
  { for(s=0,j=0;j<k;c=in.get()&1,s^=G[j++]*c);
  for(l=1<<(n-1);l;l>>=1) c=!!(l&s)+h,ou.put(c);
  }
 else
  { det.seekg(0,ios::end);N=det.tellg()/sizeof(long);det.seekg(0,ios::beg);
  for(i=0;i<N;i++)
   { det.read((char*)&u,sizeof(long)); in.seekg(u*k);
   for(s=0,j=0;j<k;c=in.get()&1,s^=G[j++]*c);
   for(l=1<<(n-1);l;l>>=1) c=!!(l&s)+h,ou.put(c);
   }
  det.close();
  }
ou.close(); in.close(); delete [] G; delete [] p; return 0;
}
```

Листинг помехоустойчивого декодера

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main(int mn,char* nm[])
{ int size,i,j,k,n,r,l,dmin,icr=0,N; char h,D=0,c=0; long long s,w;
char *p,*d=NULL; long long *G,*er,*H; int* si;
if(mn==5||mn==4); else { cerr<<nm[0]<<" in.ct out.ct p.txt <r.lb(ib)>\n"; return 1; }
ifstream det; fstream ou; ifstream cr(nm[2]); icr=!!cr; if(icr) cr.close(); else {
ou.open(nm[2],ios::out); ou.close(); }
ifstream pk(nm[3],ios::binary); if(!pk) { cerr<<"input file \""<<nm[3]<<"\" not open!\n";
return 1; }
ou.open(nm[2],ios::binary|ios::in|ios::out); if(!ou) { cerr<<"output file \""<<nm[2]<<"\"
not open!\n"; return 1; }
ifstream in(nm[1],ios::binary); if(!in) { cerr<<"input file \""<<nm[1]<<"\" not open!\n";
return 1; }
h=0x30*((nm[2][strlen(nm[2])-1]|0x20)=='t');
if(mn==5) D=((nm[4][strlen(nm[2])-2]|0x20)=='i');
in.seekg(0,ios::end);N=in.tellg();in.seekg(0,ios::beg);
pk.seekg(0,ios::end);size=pk.tellg();pk.seekg(0,ios::beg);
p=new char[size+2]; p[size]=0x0A;p[size+1]=0; pk.read(p,size); pk.close();
for(j=0,i=-1;p[++i];) if((p[i]|1)==0x31||p[i]==0x0A) p[j++]=p[i]; p[j]=0;
for(j=0,i=-1;p[++i];c=p[i]) if(p[i]==0x0A&&c==0x0A); else p[j++]=p[i]; p[j]=0;
cout<<p<<endl;
for(r=0;p[r++]!=0x0A;); k=j/r--; n=k+r;
cout<<"(n,k)=("<<n<<","<<k<<") r="<<r<<endl;
if(D) cout<<"detect\n"; else { cout<<"correct"; if((1<<r)!=n+1) cout<<" and detect";
cout<<endl; }
G=new long long[k]; memset(G,0,k*sizeof(long long));
si=new int[n];
er=new long long[n];
for(l=i=0;i<k;l++,G[i++]>>=1)
{ for(j=0;j<k;G[i]<<=1) G[i]|=(i==j++); for(j=-1;p[++j]!=0x0A;G[i]<<=1) G[i]|=p[l++]&1; }
for(dmin=n,i=1;i<1<<k;i++) { for(l=s=0,j=i;j;j>>=1) s^=G[l++]*!!(j&1); for(l=0;s;s>>=1)
l+=!!(s&1); if(dmin>l) dmin=l; }
if(mn==5) det.open(nm[4],ios::binary);
H=new long long[n]; cout<<"dmin="<<dmin<<endl;
for(i=0;i<k;i++) H[i]=G[i]&((1<<r)-1);
for(j=1<<r;j>>=1;) H[i++]=j;
for(w=1<<(n-1),j=0;j<n;si[j]=H[j],er[j++]=w,w>>=1);
for(N/=n,i=0;i<N;i++)
 { for(w=0,l=j=0;j<n;c=in.get()&1,l^=H[j++]*c,w<<=1,w|=c);
 if(!D) if(l) for(j=0;j<n;j++) if(l==si[j]) { w^=er[j]; l=0; break; }
 l=!!l; w*=!l; l<<=1; w>>=r;
 if(cr&&det) det.read((char*)&s,sizeof(long)),ou.seekg(s*k);
```

```
 for(j=1<<(k-1);j;j>>=1) c=l+!!(j&w)+h,ou.put(c);
 }
ou.close(); in.close(); if(cr&&det) det.close();
if(mn==5)
 { ifstream out(nm[2],ios::binary); ofstream det(nm[4],ios::binary);
out.seekg(0,ios::end);N=out.tellg();out.seekg(0,ios::beg);
 for(  l=s=0;s<N/k;s++) { out.seekg(s*k); c=out.get()-h; if(c==2)
l=1,det.write((char*)&s,sizeof(long)); }
 det.close(); out.close(); if(!l) remove(nm[4]);
 }
if(d) delete [] d; delete [] G; delete [] H; delete [] p; delete [] si; delete [] er;
return 0;
}
```

Канал связи, моделирующий ошибки с заданной вероятностью p:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cstring>
using namespace std;
int main(int mn,char* nm[])
{ int i,N; char c,h; srand(time(NULL));
if(mn!=4) { cerr<<nm[0]<<" in.ct out.ct Perr\n"; return 1; }
ifstream in(nm[1],ios::binary); if(!in) { cerr<<"input file \""<<nm[1]<<"\" not open!\n";
return 1; }
ofstream ou(nm[2],ios::binary); if(!ou) { cerr<<"output file \""<<nm[2]<<"\" not open!\n";
return 1; }
float perr=atof(nm[3]); if(perr<0||perr>1) { cerr<<"error Perr\n"; return 1; }
long p=RAND_MAX*perr;
in.seekg(0,ios::end);N=in.tellg();in.seekg(0,ios::beg);
h=0x30*((nm[2][strlen(nm[2])-1]|0x20)=='t');
for(i=0;i<N;i++,c=in.get()&1,c^=(rand()<p),ou.put(c+h));
return 0;
}
```

Сравнивающее устройство для информационных файлов формата «ct»

```
#include<iostream>
#include<fstream>
using namespace std;
void main(int mn,char** nm)
{ char c1,c2;
if(mn!=4) cerr<<"cmp2 inf.ct mod.fb h2\a\a"<<endl,exit(1);
ifstream in1(nm[1],ios::binary);
if(!in1) cerr<<"Файл "<<nm[1]<<" не найден!\a\a"<<endl,exit(1);
in1.seekg(0,ios::end);
long N=in1.tellg();
in1.seekg(0,ios::beg);
ifstream in2(nm[2],ios::binary);
if(!in2) cerr<<"Файл "<<nm[1]<<" не найден!\a\a"<<endl,exit(1);
float h2=atof(nm[3]);
long err=0,cl=0;
for(long i=0;i<N;i++)
 { c1=in1.get(); c2=in2.get();
 if(c1=='2'||c2=='2') cl++; else err+=c1^c2;
 }
cout<<h2<<'\t'<<err<<'\t'<<N<<'\t'<<double(err)/N<<endl;
cout<<"clear blocks="<<cl<<"\tDcl="<<double(cl)/N<<endl;
}
```

Для осуществления обратной связи, если имеется в наличии режим исправления и обнаружения ошибок, ниже приведён командный файл в среде Linux:

```
touch 2.lb
./gen 1.ct 100 0.5
while
   ./linkod 1.ct 2.ct g3.txt 1.lb
   ./chan 2.ct 3.ct 0.1
   ./lindek 3.ct 4.ct g3.txt 1.lb
   [ -e 1.lb ]
do
   cat 2.lb 1.lb > 2.lb.tmp
   mv 2.lb.tmp 2.lb
done
cmp2 1.ct 4.ct
```

Аналогичный файл в среде Windows:

```
set p=0.1
set ext=lb
if exist 1.%ext% del 1.%ext%
copy nul 2.%ext%
rem gen ver.txt 1.ct 48 1000000
:b
coder 1.ct 2.ct g3.txt 1.%ext%
chan 2.ct 3.ct %p%
decoder 3.ct 4.ct g3.txt 1.%ext%
copy 2.%ext%+1.%ext% /b
if exist 1.%ext% goto b
cmp2 1.ct 4.ct %p%
```

Замена 2-й строки на

```
set ext=ib
```

(в командном файле Linux все расширения `lb` заменить на `ib`) переводит в режим обнаружения.

Задание 4.1

Реализовать программным способом код Хемминга (7,4). Декодер должен работать и в режиме исправления ошибок и в режиме обнаружения ошибок. Для этого в программе требуется предусмотреть возможность переключения режима работы. Статистическим моделированием подтвердить теоретические формулы помехоустойчивости для этих режимов.

Варианты порождающих матриц для блочного кода Хемминга (7,4) приведены на рис. 4.6

1	1	0	0	0	0	1	1
	0	1	0	0	1	0	1
	0	0	1	0	1	1	0
	0	0	0	1	1	1	1
2	1	0	0	0	0	1	1
	0	1	0	0	1	0	1
	0	0	1	0	1	1	1
	0	0	0	1	1	1	0
3	1	0	0	0	0	1	1
	0	1	0	0	1	1	0
	0	0	1	0	1	0	1
	0	0	0	1	1	1	1
4	1	0	0	0	0	1	1
	0	1	0	0	1	1	0
	0	0	1	0	1	1	1
	0	0	0	1	1	0	1
5	1	0	0	0	0	1	1
	0	1	0	0	1	1	1
	0	0	1	0	1	0	1
	0	0	0	1	1	1	0
6	1	0	0	0	0	1	1
	0	1	0	0	1	1	1
	0	0	1	0	1	1	0
	0	0	0	1	1	0	1

7	1	0	0	0	1	0	1
	0	1	0	0	0	1	1
	0	0	1	0	1	1	0
	0	0	0	1	1	1	1
8	1	0	0	0	1	0	1
	0	1	0	0	0	1	1
	0	0	1	0	1	1	0
	0	0	0	1	1	1	1
9	1	0	0	0	1	0	1
	0	1	0	0	1	1	0
	0	0	1	0	0	1	1
	0	0	0	1	1	1	1
10	1	0	0	0	1	0	1
	0	1	0	0	1	1	0
	0	0	1	0	1	1	1
	0	0	0	1	0	1	1
11	1	0	0	0	1	0	1
	0	1	0	0	1	1	1
	0	0	1	0	0	1	1
	0	0	0	1	1	1	0
12	1	0	0	0	1	0	1
	0	1	0	0	1	1	1
	0	0	1	0	1	1	0
	0	0	0	1	0	1	1

13	1	0	0	0	1	1	0
	0	1	0	0	0	1	1
	0	0	1	0	1	0	1
	0	0	0	1	1	1	1
14	1	0	0	0	1	1	0
	0	1	0	0	0	1	1
	0	0	1	0	1	1	1
	0	0	0	1	1	0	1
15	1	0	0	0	1	1	0
	0	1	0	0	1	0	1
	0	0	1	0	0	1	1
	0	0	0	1	1	1	1
16	1	0	0	0	1	1	0
	0	1	0	0	1	0	1
	0	0	1	0	1	1	1
	0	0	0	1	0	1	1
17	1	0	0	0	1	1	0
	0	1	0	0	1	1	1
	0	0	1	0	0	1	1
	0	0	0	1	1	0	1
18	1	0	0	0	1	1	0
	0	1	0	0	1	1	1
	0	0	1	0	1	0	1
	0	0	0	1	0	1	1

19	1	0	0	0	1	1	1
	0	1	0	0	0	1	1
	0	0	1	0	1	0	1
	0	0	0	1	1	1	0
20	1	0	0	0	1	1	1
	0	1	0	0	0	1	1
	0	0	1	0	1	1	0
	0	0	0	1	1	0	1
21	1	0	0	0	1	1	1
	0	1	0	0	1	0	1
	0	0	1	0	0	1	1
	0	0	0	1	1	1	0
22	1	0	0	0	1	1	1
	0	1	0	0	1	0	1
	0	0	1	0	1	1	0
	0	0	0	1	0	1	1
23	1	0	0	0	1	1	1
	0	1	0	0	1	1	0
	0	0	1	0	0	1	1
	0	0	0	1	1	0	1
24	1	0	0	0	1	1	1
	0	1	0	0	1	1	0
	0	0	1	0	1	0	1
	0	0	0	1	0	1	1

Рис. 4.6

Вероятность ошибки декодирования систематического линейного блочного кода Хемминга (7,4), работающего в режиме исправления ошибки

$$P_{(7,4)}(\text{ош}) = 9p^2 - 26p^3 + 30p^4 - 12p^5,$$

где p — вероятность ошибки в канале связи (вероятность ошибки демодуляции).

В режиме обнаружения доля стёртых блоков при передаче всего сообщения равна:

$$D_{(7,4)}(\text{ст.бл.}) = 7p(1-p)^6 + 21p^2(1-p)^5 + 28p^3(1-p)^4 + 28p^4(1-p)^3 + 21p^5(1-p)^2 + 7p^6(1-p),$$

причём. $D_{(7,4)}(\text{ст.бл.}) < 1$

Для всех стёртых блоков необходимо сделать перезапрос. При этом, с каждой новой итерацией (очередная передача всех стёртых блоков) доля стёртых блоков меняется. Так как количество стёртых блоков конечно, то спустя некоторое количество итераций количество стёртых блоков будет сведено к нулю. Когда число стёртых блоков на очередной итерации остаётся в пределах несколько единиц или один при $D_{(7,4)}(\text{ст.бл.})$ близким к 1, то это количество может некоторое число итераций оставаться неизменным, но в итоге исход также будет сведён к нулю.

С каждой новой итерацией число ошибок будет добавляться к общему числу ошибок и поэтому на основании суммы бесконечного числа членов геометрической прогрессии $S = \frac{b_1}{1-q}$ можно найти итоговую вероятность ошибки

$$P(\text{ош}) = \frac{P_{1\text{пр.}}(\text{ош})}{1 - D_{1\text{пр.}}},$$

где $D_{1\text{пр.}}$ — доля стёртых блоков за одну итерацию.

С каждой новой итераций также будет увеличиваться длина передаваемого сообщения, что снижает общую скорость передачи данных. То есть, если сравнивать помехоустойчивость кода в режиме обнаружения с режимом исправления, то необходимо учитывать это увеличение.

График помехоустойчивости для кода Хемминга (7,4) (зависимость вероятности ошибки декодирования от вероятности ошибки в канале связи), кода декодер работает в режиме исправления ошибок, приведён на рис. 4.7.

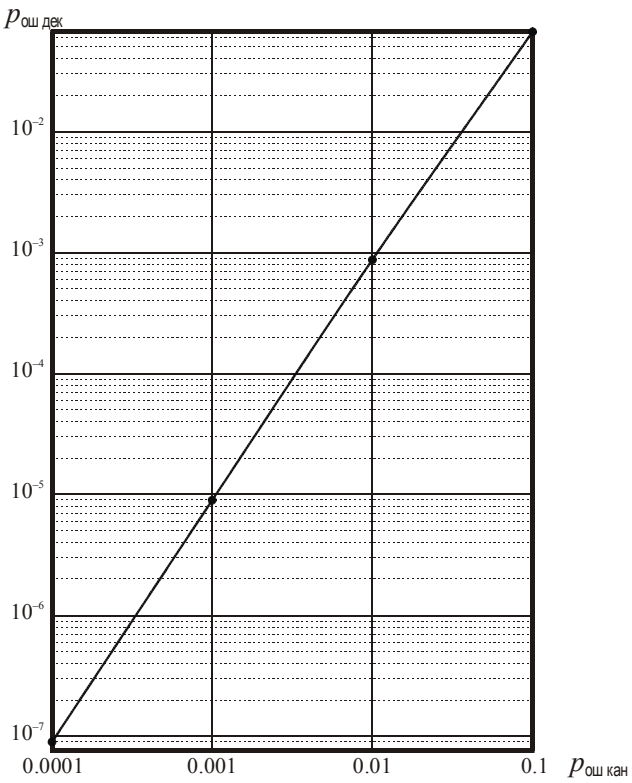


Рис. 4.7

Задание 4.2

Реализовать программным способом линейный код (8,4). Декодер должен работать и в режиме исправления ошибок и в режиме обнаружения ошибок. Для этого в программе требуется предусмотреть возможность переключения режима работы.

Варианты порождающих матриц для линейного кода (8,4) приведены на рис. 4.8

1	1	0	0	0	0	1	1	1
	0	1	0	0	1	0	1	1
	0	0	1	0	1	1	0	1
	0	0	0	1	1	1	1	0
2	1	0	0	0	0	1	1	1
	0	1	0	0	1	0	1	1
	0	0	1	0	1	1	1	0
	0	0	0	1	1	1	0	1
3	1	0	0	0	0	1	1	1
	0	1	0	0	1	1	0	1
	0	0	1	0	1	0	1	1
	0	0	0	1	1	1	1	0
4	1	0	0	0	0	1	1	1
	0	1	0	0	1	1	0	1
	0	0	1	0	1	1	1	0
	0	0	0	1	1	0	1	1
5	1	0	0	0	0	1	1	1
	0	1	0	0	1	1	1	0
	0	0	1	0	1	0	1	1
	0	0	0	1	1	1	0	1
6	1	0	0	0	0	1	1	1
	0	1	0	0	1	1	1	0
	0	0	1	0	1	1	0	1
	0	0	0	1	1	0	1	1

7	1	0	0	0	1	0	1	1
	0	1	0	0	0	1	1	1
	0	0	1	0	1	1	0	1
	0	0	0	1	1	1	1	0
8	1	0	0	0	1	0	1	1
	0	1	0	0	0	1	1	1
	0	0	1	0	1	1	1	0
	0	0	0	1	1	1	0	1
9	1	0	0	0	1	0	1	1
	0	1	0	0	1	1	0	1
	0	0	1	0	0	1	1	1
	0	0	0	1	1	1	1	0
10	1	0	0	0	1	0	1	1
	0	1	0	0	1	1	0	1
	0	0	1	0	1	1	1	0
	0	0	0	1	0	1	1	1
11	1	0	0	0	1	0	1	1
	0	1	0	0	1	1	1	0
	0	0	1	0	0	1	1	1
	0	0	0	1	1	1	0	1
12	1	0	0	0	1	0	1	1
	0	1	0	0	1	1	1	0
	0	0	1	0	1	1	0	1
	0	0	0	1	0	1	1	1

13	1	0	0	0	1	1	0	1
	0	1	0	0	0	1	1	1
	0	0	1	0	1	0	1	1
	0	0	0	1	1	1	1	0
14	1	0	0	0	1	1	0	1
	0	1	0	0	0	1	1	1
	0	0	1	0	1	1	1	0
	0	0	0	1	1	0	1	1
15	1	0	0	0	1	1	0	1
	0	1	0	0	1	0	1	1
	0	0	1	0	0	1	1	1
	0	0	0	1	1	1	1	0
16	1	0	0	0	1	1	0	1
	0	1	0	0	1	0	1	1
	0	0	1	0	1	1	1	0
	0	0	0	1	0	1	1	1
17	1	0	0	0	1	1	0	1
	0	1	0	0	1	1	1	0
	0	0	1	0	0	1	1	1
	0	0	0	1	1	0	1	1
18	1	0	0	0	1	1	0	1
	0	1	0	0	1	1	1	0
	0	0	1	0	1	0	1	1
	0	0	0	1	0	1	1	1

19	1	0	0	0	1	1	1	0
	0	1	0	0	0	1	1	1
	0	0	1	0	1	0	1	1
	0	0	0	1	1	1	0	1
20	1	0	0	0	1	1	1	0
	0	1	0	0	0	1	1	1
	0	0	1	0	1	1	0	1
	0	0	0	1	1	0	1	1
21	1	0	0	0	1	1	1	0
	0	1	0	0	1	0	1	1
	0	0	1	0	0	1	1	1
	0	0	0	1	1	1	0	1
22	1	0	0	0	1	1	1	0
	0	1	0	0	1	0	1	1
	0	0	1	0	1	1	0	1
	0	0	0	1	0	1	1	1
23	1	0	0	0	1	1	1	0
	0	1	0	0	1	1	0	1
	0	0	1	0	0	1	1	1
	0	0	0	1	1	0	1	1
24	1	0	0	0	1	1	1	0
	0	1	0	0	1	1	0	1
	0	0	1	0	1	0	1	1
	0	0	0	1	0	1	1	1

Рош1реж.испр.=28*p^3-133*p^4+280*p^5-322*p^6+204*p^7-56*p^8

Дст.бл.реж.испр.=28*p^2-168*p^3+476*p^4-784*p^5+784*p^6-448*p^7+112*p^8

Рош1реж.обн.=7*p^4-28*p^5+42*p^6-28*p^7+8*p^8

Дст.бл.реж.обн.=8*p^1-28*p^2+56*p^3-84*p^4+112*p^5-112*p^6+64*p^7-16*p^8

Задание 4.3

Реализовать программным способом линейный код согласно варианту из табл 4.1. Декодер должен работать и в режиме исправления ошибок и в режиме обнаружения ошибок. Для этого в программе требуется предусмотреть возможность переключения режима работы.

Таблица 4.1. Варианты блоковых кодов.

1) (n,k)=(7,2) r=5
Коэффициенты порождающей матрицы:
1000111
0101011
Рош1реж.испр.=8*p^3-30*p^4+48*p^5-38*p^6+12*p^7
Дст.бл.реж.испр.=21*p^2-82*p^3+150*p^4-156*p^5+92*p^6-24*p^7
Рош1реж.обн.=2*p^4-6*p^5+6*p^6-2*p^7
Дст.бл.реж.обн.=7*p-21*p^2+35*p^3-38*p^4+30*p^5-16*p^6+4*p^7

2) (n,k)=(8,2) r=6
Коэффициенты порождающей матрицы:
10001111
01110011
Рош1реж.испр.=10*p^3-30*p^4+42*p^5-30*p^6+9*p^7
Дст.бл.реж.испр.=36*p^3-135*p^4+198*p^5-135*p^6+36*p^7
Рош1реж.обн.=p^5-2*p^6+p^7
Дст.бл.реж.обн.=8*p-28*p^2+56*p^3-70*p^4+54*p^5-23*p^6+4*p^7

3) (n,k)=(8,3) r=5
Коэффициенты порождающей матрицы:
10000111
01001011
00101101
Рош1реж.испр.=16*p^3-76*p^4+160*p^5-184*p^6+112*p^7-28*p^8
Дст.бл.реж.испр.=28*p^2-140*p^3+343*p^4-504*p^5+462*p^6-244*p^7+56*p^8
Рош1реж.обн.=4*p^4-16*p^5+24*p^6-16*p^7+4*p^8
Дст.бл.реж.обн.=8*p-28*p^2+56*p^3-77*p^4+84*p^5-70*p^6+36*p^7-8*p^8

4) (n,k)=(9,2) r=7
Коэффициенты порождающей матрицы:
100011111
011100111
Рош1реж.испр.=30*p^4-138*p^5+290*p^6-336*p^7+210*p^8-56*p^9
Дст.бл.реж.испр.=84*p^3-423*p^4+963*p^5-1275*p^6+1044*p^7-504*p^8+112*p^9
Рош1реж.обн.=2*p^6-6*p^7+6*p^8-2*p^9
Дст.бл.реж.обн.=9*p-36*p^2+84*p^3-126*p^4+126*p^5-87*p^6+45*p^7-18*p^8+4*p^9

5) (n,k)=(9,3) r=6
Коэффициенты порождающей матрицы:
100000111
010001011
001001101
Рош1реж.испр.=16*p^3-92*p^4+240*p^5-360*p^6+320*p^7-156*p^8+32*p^9
Дст.бл.реж.испр.=36*p^2-196*p^3+539*p^4-924*p^5+1050*p^6-776*p^7+336*p^8-64*p^9
Рош1реж.обн.=4*p^4-20*p^5+40*p^6-40*p^7+20*p^8-4*p^9
Дст.бл.реж.обн.=9*p-36*p^2+84*p^3-133*p^4+161*p^5-154*p^6+106*p^7-44*p^8+8*p^9

6) (n,k)=(9,4) r=5
Коэффициенты порождающей матрицы:
100000111
010001011
001001101
000101110
Рош1реж.испр.=28*p^3-161*p^4+420*p^5-630*p^6+568*p^7-288*p^8+64*p^9
Дст.бл.реж.испр.=36*p^2-224*p^3+700*p^4-1344*p^5+1680*p^6-1344*p^7+624*p^8-128*p^9
Рош1реж.обн.=7*p^4-35*p^5+70*p^6-70*p^7+36*p^8-8*p^9
Дст.бл.реж.обн.=9*p-36*p^2+84*p^3-140*p^4+196*p^5-224*p^6+176*p^7-80*p^8+16*p^9

7) (n,k)=(9,5) r=4
Коэффициенты порождающей матрицы:
100000011
010000101
001000110
000100111
000011001
Рош1реж.испр.=7*p^2-28*p^3+53*p^4-51*p^5+22*p^6+p^7-3*p^8
Дст.бл.реж.испр.=12*p^2-48*p^3+84*p^4-72*p^5+24*p^6
Рош1реж.обн.=2*p^3-8*p^4+12*p^5-6*p^6-p^7+p^8
Дст.бл.реж.обн.=9*p-36*p^2+76*p^3-88*p^4+52*p^5-12*p^6

Продолжение таблицы 4.1.

8) (n,k)=(10,2) r=8
Коэффициенты порождающей матрицы:
1000011111
0101100111
Рош1реж.испр.=30*p^4-168*p^5+440*p^6-672*p^7+618*p^8-320*p^9+72*p^10
Дст.бл.реж.испр.=120*p^3-675*p^4+1764*p^5-2760*p^6+2808*p^7-1872*p^8+760*p^9-144*p^10
Рош1реж.обн.=2*p^6-8*p^7+12*p^8-8*p^9+2*p^10
Дст.бл.реж.обн.=10*p-45*p^2+120*p^3-210*p^4+252*p^5-213*p^6+132*p^7-63*p^8+22*p^9-4*p^10

9) (n,k)=(10,3) r=7
Коэффициенты порождающей матрицы:
1000001111
0100110011
0011010101
Рош1реж.испр.=10*p^3-35*p^4+59*p^5-58*p^6+32*p^7-8*p^8
Дст.бл.реж.испр.=90*p^3-480*p^4+1125*p^5-1462*p^6+1104*p^7-456*p^8+80*p^9
Рош1реж.обн.=p^5-3*p^6+3*p^7-p^8
Дст.бл.реж.обн.=10*p-45*p^2+120*p^3-210*p^4+249*p^5-198*p^6+101*p^7-30*p^8+4*p^9

10) (n,k)=(10,4) r=6
Коэффициенты порождающей матрицы:
1000000111
0100001011
0010001101
0001001110
Рош1реж.испр.=28*p^3-189*p^4+588*p^5-1085*p^6+1268*p^7-926*p^8+388*p^9-72*p^10
Дст.бл.реж.испр.=45*p^2-296*p^3+1008*p^4-2184*p^5+3220*p^6-3248*p^7+2144*p^8-832*p^9+144*p^10
Рош1реж.обн.=7*p^4-42*p^5+105*p^6-140*p^7+106*p^8-44*p^9+8*p^10
Дст.бл.реж.обн.=10*p-45*p^2+120*p^3-224*p^4+336*p^5-420*p^6+400*p^7-256*p^8+96*p^9-16*p^10

11) (n,k)=(10,5) r=5
Коэффициенты порождающей матрицы:
1000000111
0100001011
0010001101
0001001110
0000110011
Рош1реж.испр.=28*p^3-189*p^4+618*p^5-1231*p^6+1595*p^7-1328*p^8+651*p^9-144*p^10
Дст.бл.реж.испр.=45*p^2-312*p^3+1116*p^4-2568*p^5+4072*p^6-4480*p^7+3280*p^8-1440*p^9+288*p^10
Рош1реж.обн.=7*p^4-42*p^5+109*p^6-156*p^7+133*p^8-66*p^9+15*p^10
Дст.бл.реж.обн.=10*p-45*p^2+120*p^3-228*p^4+360*p^5-488*p^6+512*p^7-368*p^8+160*p^9-32*p^10

12) (n,k)=(10,6) r=4
Коэффициенты порождающей матрицы:
1000000011
0100000101
0010000110
0001000111
0000101001
0000011010
Рош1реж.испр.=8*p^2-36*p^3+86*p^4-121*p^5+101*p^6-48*p^7+13*p^8-4*p^9+p^10
Дст.бл.реж.испр.=15*p^2-74*p^3+162*p^4-186*p^5+108*p^6-24*p^7
Рош1реж.обн.=2*p^3-8*p^4+12*p^5-3*p^6-12*p^7+14*p^8-6*p^9+p^10
Дст.бл.реж.обн.=10*p-45*p^2+110*p^3-156*p^4+126*p^5-52*p^6+8*p^7

13) (n,k)=(11,2) r=9
Коэффициенты порождающей матрицы:
10000111111
01111000111
Рош1реж.испр.=35*p^4-168*p^5+392*p^6-544*p^7+462*p^8-224*p^9+48*p^10
Дст.бл.реж.испр.=260*p^4-1456*p^5+3528*p^6-4720*p^7+3668*p^8-1568*p^9+288*p^10
Рош1реж.обн.=p^7-3*p^8+3*p^9-p^10
Дст.бл.реж.обн.=11*p-55*p^2+165*p^3-330*p^4+462*p^5-462*p^6+328*p^7-158*p^8+46*p^9-6*p^10

Продолжение таблицы 4.1.

14) (n,k)=(11,3) r=8
Коэффициенты порождающей матрицы:
10000011111
01001100111
00110101011
Рош1реж.испр.=45*p^4-297*p^5+958*p^6-1887*p^7+2388*p^8-1909*p^9+882*p^10-180*p^11
Дст.бл.реж.испр.=165*p^3-1080*p^4+3366*p^5-6508*p^6+8592*p^7-7968*p^8+5054*p^9-1980*p^10+360*p^11
Рош1реж.обн.=3*p^6-15*p^7+31*p^8-33*p^9+18*p^10-4*p^11
Дст.бл.реж.обн.=11*p-55*p^2+165*p^3-330*p^4+462*p^5-468*p^6+360*p^7-226*p^8+118*p^9-44*p^10+8*p^11

15) (n,k)=(11,4) r=7
Коэффициенты порождающей матрицы:
10000001111
01000110011
00101010101
00011101010
Рош1реж.испр.=25*p^3-135*p^4+375*p^5-667*p^6+787*p^7-600*p^8+268*p^9-53*p^10
Дст.бл.реж.испр.=105*p^3-630*p^4+1668*p^5-2460*p^6+2124*p^7-1014*p^8+208*p^9
Рош1реж.обн.=2*p^5-9*p^6+16*p^7-14*p^8+6*p^9-p^10
Дст.бл.реж.обн.=11*p-55*p^2+165*p^3-330*p^4+456*p^5-432*p^6+268*p^7-98*p^8+16*p^9

16) (n,k)=(11,5) r=6
Коэффициенты порождающей матрицы:
10000000111
01000001011
00100001101
00010001110
00001010011
Рош1реж.испр.=28*p^3-217*p^4+814*p^5-1891*p^6+2936*p^7-3083*p^8+2118*p^9-865*p^10+160*p^11
Дст.бл.реж.испр.=55*p^2-402*p^3+1548*p^4-3912*p^5+7000*p^6-9040*p^7+8272*p^8-5088*p^9+1888*p^10-320*p^11
Рош1реж.обн.=7*p^4-49*p^5+151*p^6-265*p^7+289*p^8-199*p^9+81*p^10-15*p^11
Дст.бл.реж.обн.=11*p-55*p^2+165*p^3-348*p^4+588*p^5-848*p^6+1000*p^7-880*p^8+528*p^9-192*p^10+32*p^11

17) (n,k)=(11,6) r=5
Коэффициенты порождающей матрицы:
10000000111
01000001011
00100001101
00010001110
00001010011
00000110101
Рош1реж.испр.=37*p^3-287*p^4+1118*p^5-2740*p^6+4526*p^7-5082*p^8+3739*p^9-1631*p^10+320*p^11
Дст.бл.реж.испр.=55*p^2-434*p^3+1796*p^4-4904*p^5+9520*p^6-13344*p^7+13232*p^8-8800*p^9+3520*p^10-640*p^11
Рош1реж.обн.=9*p^4-63*p^5+202*p^6-380*p^7+454*p^8-346*p^9+155*p^10-31*p^11
Дст.бл.реж.обн.=11*p-55*p^2+165*p^3-356*p^4+644*p^5-1032*p^6+1360*p^7-1328*p^8+880*p^9-352*p^10+64*p^11

18) (n,k)=(11,7) r=4
Коэффициенты порождающей матрицы:
10000000011
01000000101
00100000110
00010000111
00001001001
00000101010
00000011011
Рош1реж.испр.=9*p^2-39*p^3+81*p^4-77*p^5+6*p^6+42*p^7-21*p^8+2*p^9-2*p^10
Дст.бл.реж.испр.=16*p^2-96*p^3+272*p^4-448*p^5+448*p^6-256*p^7+64*p^8
Рош1реж.обн.=3*p^3-15*p^4+32*p^5-33*p^6+16*p^7-8*p^8+9*p^9-4*p^10+p^11
Дст.бл.реж.обн.=11*p-55*p^2+152*p^3-252*p^4+256*p^5-160*p^6+64*p^7-16*p^8

Продолжение таблицы 4.1.

19) (n,k)=(12,2) r=10

Коэффициенты порождающей матрицы:

100001111111

011110001111

Рош1реж.испр.=112*p^5-728*p^6+2256*p^7-4214*p^8+5040*p^9-3816*p^10+1680*p^11-330*p^12

Дст.бл.реж.испр.=495*p^4-3336*p^5+10332*p^6-19224*p^7+23646*p^8-19880*p^9+11268*p^10-3960*p^11+660*p^12

Рош1реж.обн.=2*p^8-8*p^9+12*p^10-8*p^11+2*p^12

Дст.бл.реж.обн.=12*p-66*p^2+220*p^3-495*p^4+792*p^5-924*p^6+792*p^7-498*p^8+232*p^9-84*p^10+24*p^11-4*p^12

20) (n,k)=(12,3) r=9

Коэффициенты порождающей матрицы:

100000011111

010001100111

001010101011

Рош1реж.испр.=45*p^4-342*p^5+1273*p^6-2950*p^7+4553*p^8-4718*p^9+3169*p^10-1250*p^11+220*p^12

Дст.бл.реж.испр.=220*p^3-1575*p^4+5436*p^5-11758*p^6+17620*p^7-19088*p^8+14988*p^9-8182*p^10+2780*p^11-440*p^12

Рош1реж.обн.=3*p^6-18*p^7+46*p^8-64*p^9+51*p^10-22*p^11+4*p^12

Дст.бл.реж.обн.=12*p-66*p^2+220*p^3-495*p^4+792*p^5-930*p^6+828*p^7-586*p^8+344*p^9-162*p^10+52*p^11-8*p^12

21) (n,k)=(12,4) r=8

Коэффициенты порождающей матрицы:

100000000111

010000001011

001000001101

000100001110

Рош1реж.испр.=28*p^3-245*p^4+1008*p^5-2548*p^6+4320*p^7-5037*p^8+4000*p^9-2074*p^10+636*p^11-88*p^12

Дст.бл.реж.испр.=66*p^2-496*p^3+1975*p^4-5184*p^5+9716*p^6-13384*p^7+13500*p^8-9680*p^9+4656*p^10-1344*p^11+176*p^12

Рош1реж.обн.=7*p^4-56*p^5+196*p^6-392*p^7+491*p^8-396*p^9+202*p^10-60*p^11+8*p^12

Дст.бл.реж.обн.=12*p-66*p^2+220*p^3-509*p^4+904*p^5-1316*p^6+1576*p^7-1476*p^8+1008*p^9-464*p^10+128*p^11-16*p^12

22) (n,k)=(12,5) r=7

Коэффициенты порождающей матрицы:

100000000111

010000001011

001000001101

000100001110

000010010011

Рош1реж.испр.=28*p^3-245*p^4+1038*p^5-2754*p^6+4978*p^7-6284*p^8+5490*p^9-3182*p^10+1106*p^11-175*p^12

Дст.бл.реж.испр.=66*p^2-512*p^3+2115*p^4-5808*p^5+11500*p^6-16888*p^7+18312*p^8-14240*p^9+7504*p^10-2400*p^11+352*p^12

Рош1реж.обн.=7*p^4-56*p^5+200*p^6-416*p^7+554*p^8-488*p^9+280*p^10-96*p^11+15*p^12

Дст.бл.реж.обн.=12*p-66*p^2+220*p^3-513*p^4+936*p^5-1436*p^6+1848*p^7-1880*p^8+1408*p^9-720*p^10+224*p^11-32*p^12

23) (n,k)=(12,6) r=6

Коэффициенты порождающей матрицы:

100000000111

010000001011

001000001101

000100001110

000010010011

000001010101

Рош1реж.испр.=37*p^3-324*p^4+1414*p^5-3921*p^6+7468*p^7-9988*p^8+9275*p^9-5716*p^10+2106*p^11-351*p^12

Дст.бл.реж.испр.=66*p^2-544*p^3+2395*p^4-7056*p^5+15068*p^6-23896*p^7+27936*p^8-23360*p^9+13200*p^10-4512*p^11+704*p^12

Рош1реж.обн.=9*p^4-72*p^5+265*p^6-582*p^7+834*p^8-800*p^9+501*p^10-186*p^11+31*p^12

Дст.бл.реж.обн.=12*p-66*p^2+220*p^3-521*p^4+1000*p^5-1676*p^6+2392*p^7-2688*p^8+2208*p^9-1232*p^10+416*p^11-64*p^12

Продолжение таблицы 4.1.

24) (n,k)=(12,7) r=5

Коэффициенты порождающей матрицы:

100000000111
010000001011
001000001101
000100001110
000010010011
000001010101
000000110110

Рош1реж.испр.=52*p^3-455*p^4+2016*p^5-5716*p^6+11240*p^7-15696*p^8+15360*p^9-10032*p^10+3936*p^11-704*p^12

Дст.бл.реж.испр.=66*p^2-596*p^3+2850*p^4-9072*p^5+20784*p^6-35136*p^7+43632*p^8-38720*p^9+23232*p^10-8448*p^11+1408*p^12

Рош1реж.обн.=13*p^4-104*p^5+388*p^6-872*p^7+1296*p^8-1312*p^9+880*p^10-352*p^11+64*p^12

Дст.бл.реж.обн.=12*p-66*p^2+220*p^3-534*p^4+1104*p^5-2064*p^6+3264*p^7-3984*p^8+3520*p^9-2112*p^10+768*p^11-128*p^12

25) (n,k)=(12,8) r=4

Коэффициенты порождающей матрицы:

100000000011
010000000101
001000000110
000100000111
000010001001
000001001010
000000101011
000000011100

Рош1реж.испр.=12*p^2-65*p^3+188*p^4-329*p^5+353*p^6-208*p^7+30*p^8+37*p^9-21*p^10+4*p^11-p^12

Дст.бл.реж.испр.=15*p^2-99*p^3+300*p^4-504*p^5+480*p^6-240*p^7+48*p^8

Рош1реж.обн.=4*p^3-24*p^4+66*p^5-101*p^6+91*p^7-52*p^8+25*p^9-13*p^10+4*p^11

Дст.бл.реж.обн.=12*p-66*p^2+203*p^3-380*p^4+440*p^5-304*p^6+112*p^7-16*p^8

Строка (формула), выведенная ЭВМ:

$D_{(7,4)}(сб)=7*p*(1-p)^6+21*p^2*(1-p)^5+(35-7)*p^3*(1-p)^4+(35-7)*p^4*(1-p)^3+21*p^5*(1-p)^2+7*p^6*(1-p)$

Пояснения вывода формулы. Ошибки для кодовой комбинации 0000000 в коде Хемминга (7,4): 1 столбец: где произошли ошибки; 2 столбец: результат после декодирования для информационных символов (исправление в сторону неправильной разрешённой комбинации); 3 столбец: количество ошибок (как отличие от правильного результата 0000

1100000 1100 2	1110000 1110 3	1111000 0111 3	1111100 1011 3	1111110 1111 4	1111111 1111 4
1010000 1010 2	1101000 1101 3	1110100 1100 2	1111010 1101 3	1111101 1111 4	4
1001000 1001 2	1100100 1100 2	1110010 1010 2	1111001 1110 3	1111011 1111 4	
1000100 1100 2	1100010 1101 3	1110001 1110 3	1110110 0110 2	1110111 1111 4	
1000010 1010 2	1100001 1110 3	1101100 1100 2	1110101 1110 3	1101111 1111 4	
1000001 1001 2	1011000 1011 3	1101010 1101 3	1110011 1110 3	1011111 1111 4	
0110000 0111 3	1010100 1011 3	1101001 1001 2	1101110 1101 3	0111111 1111 4	
0101000 0111 3	1010010 1010 2	1100110 1100 2	1101101 0101 2	28	
0100100 1100 2	1010001 1110 3	1100101 1100 2	1101011 1101 3		
0100010 0100 1	1001100 1011 3	1100011 0100 1	1100111 1000 1		
0100001 0100 1	1001010 1101 3	1011100 1011 3	1011110 1011 3		
0011000 0111 3	1001001 1001 2	1011010 1010 2	1011101 1011 3		
0010100 0010 1	1000110 1000 1	1011001 1001 2	1011011 0011 2		
0010010 1010 2	1000101 1000 1	1010110 1010 2	1010111 1000 1		
0010001 0010 1	1000011 1000 1	1010101 0010 1	1001111 1000 1		
0001100 0001 1	0111000 0111 3	1010011 1010 2	0111110 0110 2		
0001010 0001 1	0110100 0110 2	1001110 0001 1	0111101 0101 2		
0001001 1001 2	0110010 0110 2	1001101 1001 2	0111011 0011 2		
0000110 0001 1	0110001 1110 3	1001011 1001 2	0110111 0110 2		
0000101 0010 1	0101100 0101 2	1000111 1000 1	0101111 0101 2		
0000011 0100 1	0101010 1101 3	0111100 0111 3	0011111 0011 2		
36	0101001 0101 2	0111010 0111 3	48		
	0100110 0110 2	0111001 0111 3			
	0100101 0101 2	0110110 0110 2			
	0100011 0100 1	0110101 0010 1			
	0011100 1011 3	0110011 0100 1			
	0011010 0011 2	0101110 0001 1			
	0011001 0011 2	0101101 0101 2			
	0010110 0110 2	0101011 0100 1			
	0010101 0010 1	0100111 0100 1			
	0010011 0011 2	0011110 0001 1			
	0001110 0001 1	0011101 0010 1			
	0001101 0101 2	0011011 0011 2			
	0001011 0011 2	0010111 0010 1			
	0000111 1000 1	0001111 0001 1			
	76	64			

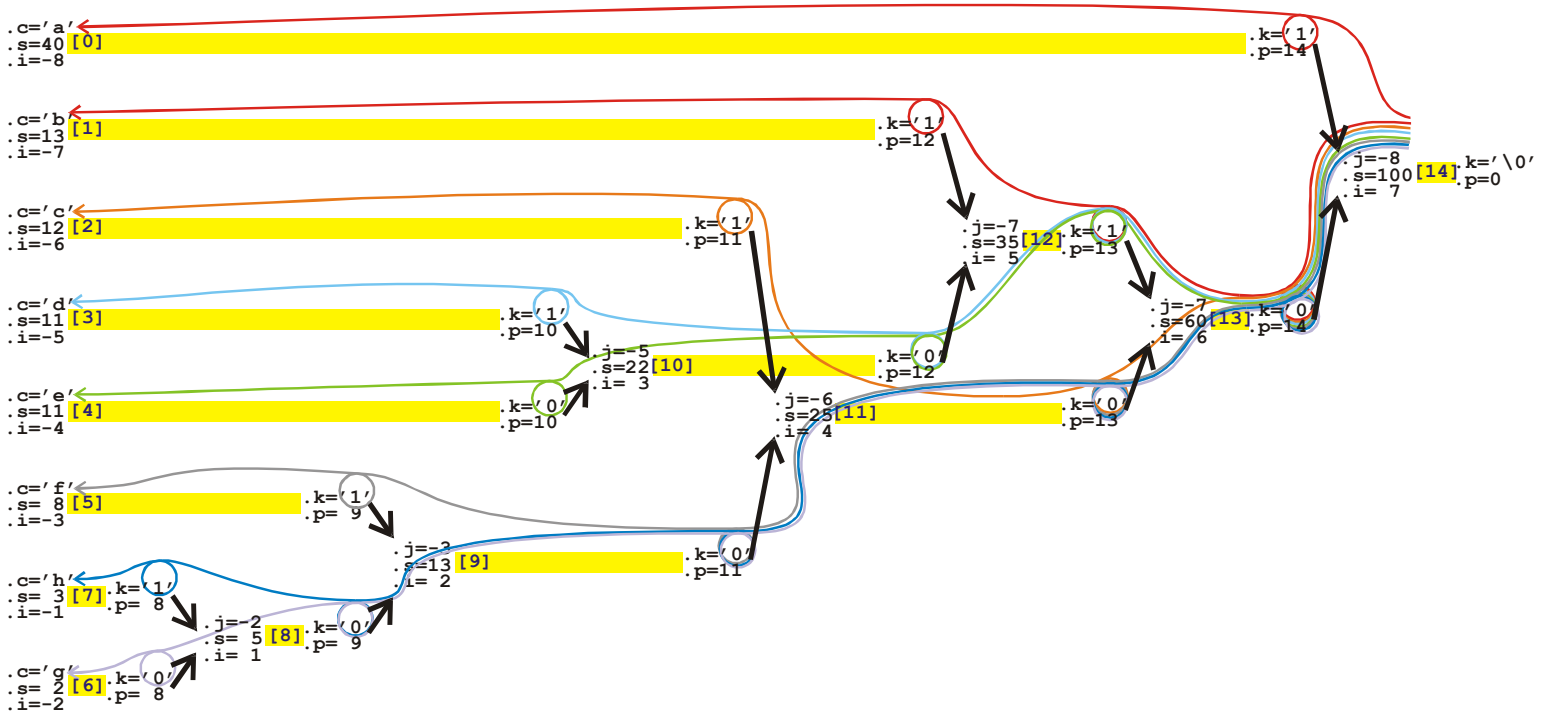
Режим исправления. Количество ошибок при одной итерации из числа переданных символов	Режим исправления. Количество стёртых блоков за одну итерацию	**Режим обнаружения Количество ошибок при одной итерации из числа переданных символов**	Режим обнаружения Количество стёртых блоков за одну итерацию	**Вероятность появления комбинации**
0: 0 4	0: 0 4	0: 0 4	0: 0 4	$p^0(1-p)^7$
1: 0 28	1: 0 28	1: 0 28	1: 7 28	$p^1(1-p)^6$
2: 36 84	2: 0 84	2: 0 84	2: 21 84	$p^2(1-p)^5$
3: 76 140	3: 0 140	3: 12 140	3: 28 140	$p^3(1-p)^4$
4: 64 140	4: 0 140	4: 16 140	4: 28 140	$p^4(1-p)^3$
5: 48 84	5: 0 84	5: 0 84	5: 21 84	$p^5(1-p)^2$
6: 28 28	6: 0 28	6: 0 28	6: 7 28	$p^6(1-p)^1$
7: 4 4	7: 0 4	7: 4 4	7: 0 4	$p^7(1-p)^0$

```
#include <fstream>
#include <iostream>
using namespace std;
void ef(char*& n) { int j=0,k=-1; for(;n[++k];); for(;--k+1;) if(n[k]=='\\') break; for(;n[++k];n[j++]=n[k]); n[j]=0;}
struct haff
{ unsigned s; // количество
int i; // индекс
int p; // ссылка
char c; // буква
char k; // цифровой код
int j; // направляющий индекс
void print() {cout<<"c="<<c<<" i="<<i<<" j="<<j<<" p="<<p<<" s="<<s<<" k="<<k<<endl;}
};
#define Q 0 // 1 - по вероятности, 0 - согласно ссылке на сортировку
int main(int mn,char* nm[])
{ int b[256],N,size,i,j,k,min,jmin1,jmin2,max,imax,Max=-2u/2,s; haff* h; unsigned char *d,*p,g,r,*v,**V;
for(ef(nm[0]),i=0;i<256;b[i++]=0); if(mn!=3) cerr<<nm[0]<<" in.ext out.arh\n",exit(1);
ifstream in(nm[1],ios::binary); if(!in) cerr<<"Файл \""<<nm[1]<<"\" не открыт!\n",exit(1);
in.seekg(0,ios::end);size=in.tellg();in.seekg(0,ios::beg);
p=new unsigned char[size]; in.read((char*)p,size);in.close();
for(i=0;i<size;i++) b[p[i]]++; // Подсчёт количества выпадения каждого из 256 символов
for(N=i=0;i<256;i++) N+=!!b[i]; // Подсчёт количества уникальных символов, содержащихся в исходном файле
cout<<"N="<<N<<endl; // вывод количества уникальных символов, содержащихся в исходном файле
h=new haff[2*N-1];if(!h) cerr<<"Error memory!\n",exit(1); // создание массива структур с учётом дополнительных узлов
                                 // при построении схемы для определения 0 и 1 цифрового кода и ссылок на узлы сходящихся ветвей
for(j=0;j<2*N-1;j++) h[j].c=h[j].s=h[j].i=h[j].k=h[j].p=h[j].j=0; // Обнуление каждого элемента структуры
                                                                  //для каждого элемента массива структур
for(j=i=0;i<256;i++) if(b[i]) h[j].s=b[i],h[j++].c=i; // запись в .s количества выпадений символа и в .c кода по ASCII самого символа
                             // для символов с ненулевым выпадением в элементы структуры, по порядку в первые элементы массива структур
unsigned char* S=new unsigned char[N];// массив под размеры кодовых последовательностей для каждого символа
for(j=-N;j<0;imax>=0?h[imax].i=h[imax].j=j++:j++) for(imax=-1,max=k=0;k<N;k++) if(!h[k].i) if(h[k].s>max) max=h[k].s,imax=k;
    //Заполнение отрицательными индексами элементов первой части массива структур с учётом сортировки по убыванию количества
     //выпадения символа в исходном файле (первый индекс самый минимальный отрицательный)
for(i=0;i<N-1;i++) // добавление ячеек
 { for(min=Max,jmin1=-1,j=0;j<N+i;j++) // от первой до последней добавленной ячейки
    if(!h[j].p) if(min>=h[j].s) min=h[j].s,jmin1=j; // поиск первого символа с минимальным количеством выпадения
                                                    // среди неучтённых символов (неучтённый символ .p==0)
 h[jmin1].p=N+i; // учёт символа записью ссылки на новый элемент из дополнительной области массива структур
 for(min=Max,jmin2=-1,j=0;j<N+i;j++) // от первой до последней добавленной ячейки
  if(!h[j].p) if(min>=h[j].s) min=h[j].s,jmin2=j; // поиск второго символа с минимальным количеством выпадения
                                                  // среди неучтённых символов
 h[jmin2].p=N+i; // учёт символа записью ссылки на новый элемент из дополнительной области массива структур
 if(Q) h[jmin1].k=0x30,h[jmin2].k=0x31; // установка цифрового кода (0 или 1) по вероятности
 else
  if(h[jmin1].j<h[jmin2].j) h[N+i].j=h[jmin1].j,h[jmin1].k=0x31,h[jmin2].k=0x30; // установка цифрового кода (0 или 1)
  else                      h[N+i].j=h[jmin2].j,h[jmin1].k=0x30,h[jmin2].k=0x31; // согласно ссылке на сортировку
 h[N+i].s=h[jmin1].s+h[jmin2].s; // определение количества выпадений для нового узлового суммарного символа
 h[N+i].i=i+1; // присвоение новому символу очередного индекса (начиная с 1, ноль - признак отсутствия индексации)
 }
```

```
ofstream ou(nm[2],ios::binary); if(!ou) cerr<<"Файл \""<<nm[2]<<"\" не создан!\n",exit(1);
for(ou.put(N-1),s=0,j=-1;++j<N;ou.put(h[j].c),ou.put(S[j]),s+=S[j]) // вывод в заголовок количества символов-1, и пар (код символа и
размер кодовой последовательности для этого символа)
 for(S[k=j]=0;h[k].p;k=h[k].p,++S[j]); // подсчёт числа кодовых бит для каждой кодовой последовательности
d=new unsigned char[s];v=new unsigned char[s]; V=new unsigned char*[N]; // в массиве d в обратном порядке, в v и V - в прямом
for(i=j=-1;++j<N;k++,i--) // запись кодовых последовательностей в обратном порядке
 for(S[k=j]=0;d[++i]=h[k].k,h[k].p;k=h[k].p) S[j]++;
for(j=-1,k=0;++j<N;) V[j]=&v[k],k+=S[j];

for(g=8,s=r=k=0,j=-1;++j<N;k+=S[j])
 for(i=S[j];i--;r<<=1)
  {
  r|=d[k+i]&1,v[s++]=d[k+i];
  if(--g); else ou.put(r),g=8,r=0; // запись кодовых последовательностей в заголовок
  }
for(;--g;r<<=1); ou.put(r); // запись в заголовок последнего байта, содержащего часть кодовой последовательности
for(j=-1,k=0;++j<N;) V[j]=&v[k],k+=S[j];
for(j=-1;++j<N;cout<<endl) for(cout<<'\''<<h[j].c<<"\' ",i=-1;++i<S[j];)  cout<<V[j][i];
for(i=0;i<256;b[i++]=-1); for(i=0;i<N;b[(unsigned char)h[i].c]=i++); // запись в b[] индексов согласно массиву структур для
определения индекса по считанному байту
for(g=k=j=0;j<size;j++) // формирование байтов из закодированного сообщения
 for(i=0;i<S[b[p[j]]];g|=(V[b[p[j]]][i++]&1)<<(7-k%8),++k%8?1:(ou.put(g),g=0));
if(k%8) ou.put(g); // последний байт, если количество нулей и единиц сжатого текста не кратно 8
cout<<"Кодированная часть файла в битах: "<<k<<endl;
g=k%8,ou.put(g); // количество нулей и единиц сжатого текста
ou.close(); delete [] V;delete [] v;delete [] d;delete [] S;delete [] p;delete [] h; return 0;
}
```

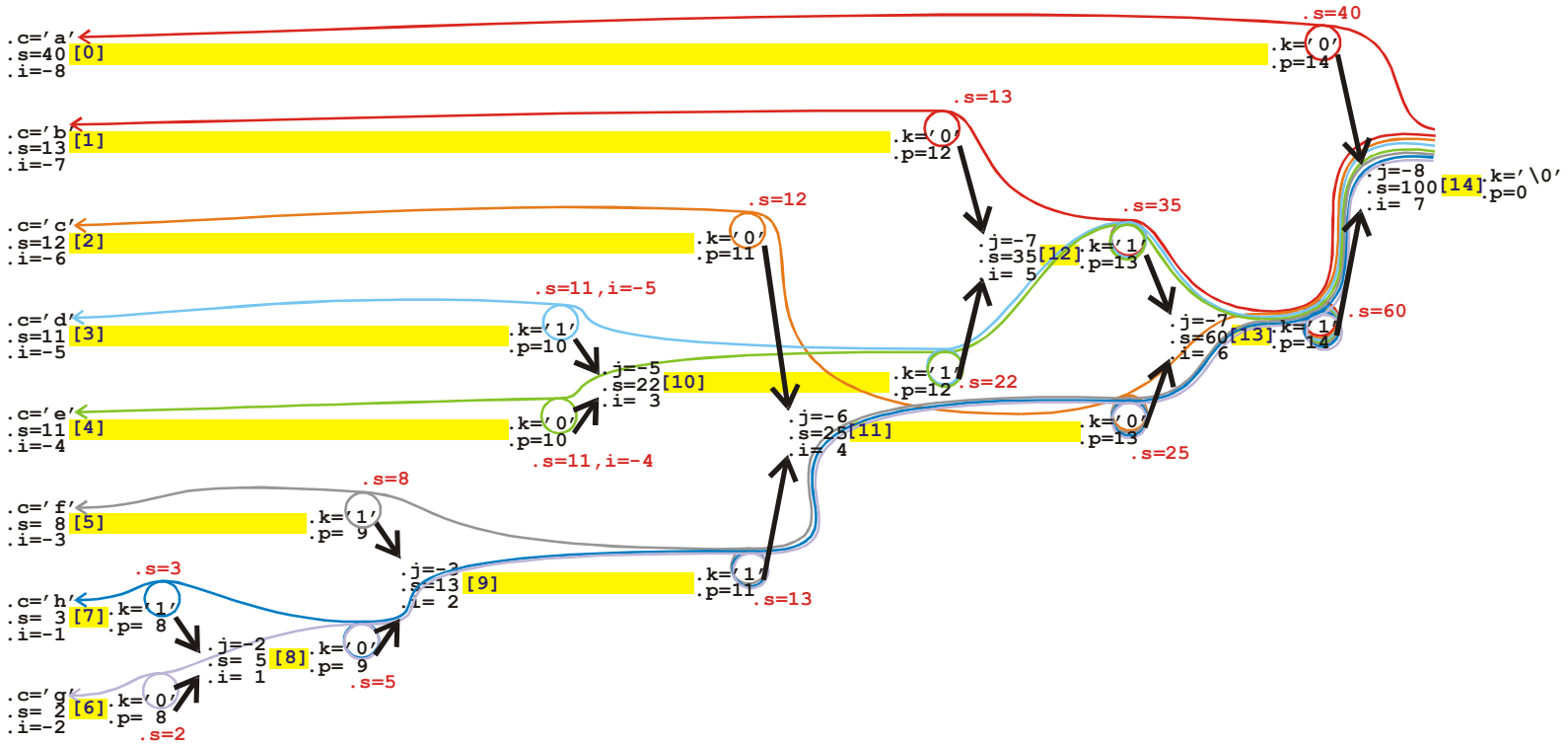
```
'a': 40     'b': 13     'c': 12     'd': 11     'e': 11     'f': 8      'g': 2      'h': 3
Установка 0 и 1 согласно порядку (индекс j копирует индекс i для обозначения ссылки на верхний уровень сортированных исходных данных)
 [0] .s= 40 .i=-8 .j=-8 .p=14 .k= '1' .c= 'a'
 [1] .s= 13 .i=-7 .j=-7 .p=12 .k= '1' .c= 'b'
 [2] .s= 12 .i=-6 .j=-6 .p=11 .k= '1' .c= 'c'
 [3] .s= 11 .i=-5 .j=-5 .p=10 .k= '1' .c= 'd'
 [4] .s= 11 .i=-4 .j=-4 .p=10 .k= '0' .c= 'e'
 [5] .s=  8 .i=-3 .j=-3 .p= 9 .k= '1' .c= 'f'
 [6] .s=  2 .i=-1 .j=-1 .p= 8 .k= '0' .c= 'g'
 [7] .s=  3 .i=-2 .j=-2 .p= 8 .k= '1' .c= 'h'
 [8] .s=  5 .i= 1 .j=-2 .p= 9 .k= '0' .c='\0'
 [9] .s= 13 .i= 2 .j=-3 .p=11 .k= '0' .c='\0'
[10] .s= 22 .i= 3 .j=-5 .p=12 .k= '0' .c='\0'
[11] .s= 25 .i= 4 .j=-6 .p=13 .k= '0' .c='\0'
[12] .s= 35 .i= 5 .j=-7 .p=13 .k= '1' .c='\0'
[13] .s= 60 .i= 6 .j=-7 .p=14 .k= '0' .c='\0'
[14] .s=100 .i= 7 .j=-8 .p= 0 .k='\0' .c='\0'
```




```
'a': 1
'b': 011
'c': 001
'd': 0101
'e': 0100
'f': 0001
'g': 00000
'h': 00001
```

Установка 0 и 1 согласно частости (при равенстве согласно порядку)

```
 [0] .s= 40 .i=-8 .j=-8 .p=14 .k= '0' .c= 'a'
 [1] .s= 13 .i=-7 .j=-7 .p=12 .k= '0' .c= 'b'
 [2] .s= 12 .i=-6 .j=-6 .p=11 .k= '0' .c= 'c'
 [3] .s= 11 .i=-5 .j=-5 .p=10 .k= '1' .c= 'd'
 [4] .s= 11 .i=-4 .j=-4 .p=10 .k= '0' .c= 'e'
 [5] .s=  8 .i=-3 .j=-3 .p= 9 .k= '1' .c= 'f'
 [6] .s=  2 .i=-1 .j=-1 .p= 8 .k= '0' .c= 'g'
 [7] .s=  3 .i=-2 .j=-2 .p= 8 .k= '1' .c= 'h'
 [8] .s=  5 .i= 1 .j= 0 .p= 9 .k= '0' .c='\0'
 [9] .s= 13 .i= 2 .j= 0 .p=11 .k= '1' .c='\0'
[10] .s= 22 .i= 3 .j= 0 .p=12 .k= '1' .c='\0'
[11] .s= 25 .i= 4 .j= 0 .p=13 .k= '0' .c='\0'
[12] .s= 35 .i= 5 .j= 0 .p=13 .k= '1' .c='\0'
[13] .s= 60 .i= 6 .j= 0 .p=14 .k= '1' .c='\0'
[14] .s=100 .i= 7 .j= 0 .p= 0 .k='\0' .c='\0'
```




```
'a': 0
'b': 110
'c': 100
'd': 1111
'e': 1110
'f': 1011
'g': 10100
'h': 10101
```

```
#include <fstream>
#include <iostream>
using namespace std;
void ef(char*& n) { int j=0,k=-1; for(;n[++k];); for(;--k+1;) if(n[k]=='\\') break; for(;n[++k];n[j++]=n[k]); n[j]=0;}
int main(int mn,char* nm[])
{ int N,size,i,j,k,l,s,r,e,R; unsigned char *p,*d,*v,**V,*a,*S,g; short *c,w;
ef(nm[0]); if(mn!=3) cerr<<nm[0]<<" in.arh out.ext\n",exit(1);
ifstream in(nm[1],ios::binary); if(!in) cerr<<"Файл \""<<nm[1]<<"\" не открыт!\n",exit(1);
in.seekg(0,ios::end);size=in.tellg();in.seekg(0,ios::beg);
p=new unsigned char[size]; in.read((char*)p,size);in.close(); N=(unsigned)p[0]+1;
a=new unsigned char[N]; // массив символов
S=new unsigned char[N]; // массив размеров кодовых последовательностей
c=new short[2*N-2]; for(i=0;i<2*N-2;c[i++]=-1); // массив индексов декодирования
cout<<"Размер таблицы равен "<<N<<" символам."<<endl; // вывод количества уникальных символов, содержащихся в исходном файле
for(s=0,j=-1;++j<N;a[j]=p[j*2+1],s+=(S[j]=p[j*2+2]));
v=new unsigned char[s]; V=new unsigned char*[N];
for(k=-1,j=N*2+1;j<s/8+!!(s%8)+N*2+1;j++) for(g=128;g;g>>=1) v[++k]=0x30+!!(g&p[j]);
for(j=-1,k=0;++j<N;) V[j]=&v[k],k+=S[j];
for(j=-1;++j<N;cout<<endl) for(cout<<'\''<<a[j]<<"\' ",i=-1;++i<S[j];)  cout<<V[j][i];
cout<<"Заголовок "<<(R=s/8+!!(s%8)+2*N+1)<<" байт + 1 служебный байт в конце файла."<<endl;
cout<<"Сжатый текст содержит  "<<(s=(size-R-2)*8+p[size-1])<<" бит в байтах количеством "<<size-R-1<<endl;
for(l=j=0;j<N;j++)
 for(k=2*N-2,i=0;i<S[j];i++) // цикл побитового сканирования начиная с последней ячейки
  { if(w=v[l++]&1,c[(k-N)*2+w]<0) // если для этого номера и кода ссылка не закреплена
     if(i==S[j]-1) c[(k-N)*2+w]=j;// если это последний код, номер ссылки на символ
     else
      { for(e=k-1;e>=N;--e) if(c[(e-N)*2+w]<0 && c[(e-N)*2+!w]<0) break; // поиск пары свободных ячеек одного уровня
      if(c[(e-N)*2+!w]<0) k=c[(k-N)*2+w]=e; // если ячейка
      else
       { for(;e>=N;--e) if(c[(e-N)*2+!w]<0 && c[(e-N)*2+w]<0) break;
       k=c[(k-N)*2+w]=e;
       }
      }
    else k=c[(k-N)*2+w]; // меняем номер по ссылке, если для этого номера и кода ссылка закреплена
  }
ofstream ou(nm[2],ios::binary); if(!ou) cerr<<"Файл \""<<nm[1]<<"\" не создан!\n",exit(1);
for(l=R,g=128,w=N-2;(l+=!g)<size-2;) // декодирование, вывод исходного текста
 { g+=128*!g,k=!!(p[l]&g),g>>=1;
 if(c[2*w+k]<N) ou.put(a[c[2*w+k]]),w=N-2;
 else w=c[2*w+k]-N;
 }
for(l=-1,g=128;++l<p[size-1];)
 { k=!!(p[size-2]&g),g>>=1;
 if(c[2*w+k]<N) ou.put(a[c[2*w+k]]),w=N-2;
 else w=c[2*w+k]-N;
 }
ou.close(); delete [] v,delete [] V,delete [] S, delete [] p, delete [] c, delete [] a; return 0;
}
```

```
Количество символов в таблице: 8
[0]: 'a': 1
[1]: 'b': 011
[2]: 'c': 001
[3]: 'd': 0101
[4]: 'e': 0100
[5]: 'f': 0001
[6]: 'g': 00001
[7]: 'h': 00000
[0]: 7           [1]: 6
[2]: 8           [3]: 5
[4]: 4           [5]: 3
[6]: 9           [7]: 2
[8]: 10          [9]: 1
[10]: 11         [11]: 12
[12]: 13         [13]: 0
Сжатый текст содержал 260 бит.
'a': 1
'b': 011
'c': 001
'd': 0101
'e': 0100
'f': 0001
'g': 00001
'h': 00000
```

Алгоритм декодирования

```
w=N-2
if(c[2*w+k]<N)
 ou.put(a[c[2*w+k]]),w=N-2;
else
 w=c[2*w+k]-N;
```

```
Заполнение таблицы ссылок
'a': 1
 [0]: -2   [1]: -2
 [2]: -2   [3]: -2
 [4]: -2   [5]: -2
 [6]: -2   [7]: -2
 [8]: -2   [9]: -2
[10]: -2  [11]: -2
[12]: -1  [13]:  0

'b': 011
 [0]: -2   [1]: -2
 [2]: -2   [3]: -2
 [4]: -2   [5]: -2
 [6]: -2   [7]: -2
 [8]: -1   [9]:  1
[10]: -1  [11]: 12
[12]: 13  [13]:  0

'c': 001
 [0]: -2   [1]: -2
 [2]: -2   [3]: -2
 [4]: -2   [5]: -2
 [6]: -1   [7]:  2
 [8]: -1   [9]:  1
[10]: 11  [11]: 12
[12]: 13  [13]:  0

'd': 0101
 [0]: -2   [1]: -2
 [2]: -2   [3]: -2
 [4]: -1   [5]:  3
 [6]: -1   [7]:  2
 [8]: 10   [9]:  1
[10]: 11  [11]: 12
[12]: 13  [13]:  0

'e': 0100
 [0]: -2   [1]: -2
 [2]: -2   [3]: -2
 [4]:  4   [5]:  3
 [6]: -1   [7]:  2
 [8]: 10   [9]:  1
[10]: 11  [11]: 12
[12]: 13  [13]:  0

'f': 0001
 [0]: -2   [1]: -2
 [2]: -1   [3]:  5
 [4]:  4   [5]:  3
 [6]:  9   [7]:  2
 [8]: 10   [9]:  1
[10]: 11  [11]: 12
[12]: 13  [13]:  0

'g': 00001
 [0]: -1   [1]:  6
 [2]:  8   [3]:  5
 [4]:  4   [5]:  3
 [6]:  9   [7]:  2
 [8]: 10   [9]:  1
[10]: 11  [11]: 12
[12]: 13  [13]:  0

'h': 00000
 [0]:  7   [1]:  6
 [2]:  8   [3]:  5
 [4]:  4   [5]:  3
 [6]:  9   [7]:  2
 [8]: 10   [9]:  1
[10]: 11  [11]: 12
[12]: 13  [13]:  0
```

```
 [0]:  7   [1]:  6
 [2]:  8   [3]:  5
 [4]:  4   [5]:  3
 [6]:  9   [7]:  2
 [8]: 10   [9]:  1
[10]: 11  [11]: 12
[12]: 13  [13]:  0

[0]: 'a': 1
[1]: 'b': 011
[2]: 'c': 001
[3]: 'd': 0101
[4]: 'e': 0100
[5]: 'f': 0001
[6]: 'g': 00001
[7]: 'h': 00000

'h' 0₄0₃0₂0₁0₀
w=N-2=8-2=6
c[2*w+k=2*6+0₄=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₃=10]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+0₂=6]=9 < 8(false)
w=c[2*w+k]-N=9-8=1
c[2*w+k=2*1+0₁=2]=8 < 8(false)
w=c[2*w+k]-N=8-8=0
c[2*w+k=2*0+0₀=0]=7 < 8(true)
[7]: 'h'

'g' 0₄0₃0₂0₁0₀
w=N-2=8-2=6
c[2*w+k=2*6+0₄=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₃=10]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+0₂=6]=9 < 8(false)
w=c[2*w+k]-N=9-8=1
c[2*w+k=2*1+0₁=2]=8 < 8(false)
w=c[2*w+k]-N=8-8=0
c[2*w+k=2*0+1₀=1]=6 < 8(true)
[6]: 'g'

'f' 0₃0₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₃=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₂=10]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+0₁=6]=9 < 8(false)
w=c[2*w+k]-N=9-8=1
c[2*w+k=2*1+1₀=3]=5 < 8(true)
[5]: 'f'

'e' 0₃1₂0₁0₀
w=N-2=8-2=6
c[2*w+k=2*6+0₃=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+1₂=11]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+0₁=8]=10 < 8(false)
w=c[2*w+k]-N=10-8=2
c[2*w+k=2*2+0₀=4]=4 < 8(true)
[4]: 'e'

'd' 0₃1₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₃=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+1₂=11]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+0₁=8]=10 < 8(false)
w=c[2*w+k]-N=10-8=2
c[2*w+k=2*2+1₀=5]=3 < 8(true)
[3]: 'd'

'c' 0₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₂=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₁=10]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+1₀=7]=2 < 8(true)
[2]: 'c'

'b' 0₂1₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₂=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+1₁=11]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+1₀=9]=1 < 8(true)
[1]: 'b'

'b' 1₀
w=N-2=8-2=6
c[2*w+k=2*6+1₂=13]=0 < 8(true)
[0]: 'a'
```

```
 [0]:  0   [1]:  1
 [2]:  8   [3]:  2
 [4]:  9   [5]:  3
 [6]: 10   [7]:  4
 [8]: 11   [9]:  5
[10]: 12  [11]:  6
[12]: 13  [13]:  7

[0]: '0' 0000000
[1]: '1' 0000001
[2]: '2' 000001
[3]: '3' 00001
[4]: '4' 0001
[5]: '5' 001
[6]: '6' 01
[7]: '7' 1
P(0) = 0.000304878      (1)
P(1) = 0.000914634      (3)
P(2) = 0.0027439        (9)
P(3) = 0.00823171       (27)
P(4) = 0.0246951        (81)
P(5) = 0.0740854        (243)
P(6) = 0.222256         (729)
P(7) = 0.666768         (2187)


'0' 0₆0₅0₄0₃0₂0₁0₀
w=N-2=8-2=6
c[2*w+k=2*6+0₆=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₅=10]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+0₄=8]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+0₃=6]=10 < 8(false)
w=c[2*w+k]-N=10-8=2
c[2*w+k=2*2+0₂=4]=9 < 8(false)
w=c[2*w+k]-N=9-8=1
c[2*w+k=2*1+0₁=2]=8 < 8(false)
w=c[2*w+k]-N=8-8=0
c[2*w+k=2*0+0₀=0]=0 < 8(true)
[0]: '0'

'1' 0₆0₅0₄0₃0₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₆=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₅=10]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+0₄=8]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+0₃=6]=10 < 8(false)
w=c[2*w+k]-N=10-8=2
c[2*w+k=2*2+0₂=4]=9 < 8(false)
w=c[2*w+k]-N=9-8=1
c[2*w+k=2*1+0₁=2]=8 < 8(false)
w=c[2*w+k]-N=8-8=0
c[2*w+k=2*0+1₀=1]=1 < 8(true)
[1]: '1'

'2' 0₅0₄0₃0₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₅=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₄=10]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+0₃=8]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+0₂=6]=10 < 8(false)
w=c[2*w+k]-N=10-8=2
c[2*w+k=2*2+0₁=4]=9 < 8(false)
w=c[2*w+k]-N=9-8=1
c[2*w+k=2*1+1₁=3]=2 < 8(true)
[2]: '2'

'3' 0₄0₃0₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₄=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₃=10]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+0₂=8]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+0₁=6]=10 < 8(false)
w=c[2*w+k]-N=10-8=2
c[2*w+k=2*2+1₀=5]=3 < 8(true)
[3]: '3'

'4' 0₃0₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₃=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₂=10]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+0₁=8]=11 < 8(false)
w=c[2*w+k]-N=11-8=3
c[2*w+k=2*3+1₀=7]=4 < 8(true)
[4]: '4'

'5' 0₂0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₂=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+0₁=10]=12 < 8(false)
w=c[2*w+k]-N=12-8=4
c[2*w+k=2*4+1₀=9]=5 < 8(true)
[5]: '5'

'6' 0₁1₀
w=N-2=8-2=6
c[2*w+k=2*6+0₁=12]=13 < 8(false)
w=c[2*w+k]-N=13-8=5
c[2*w+k=2*5+1₀=11]=6 < 8(true)
[6]: '6'

'7' 1₀
w=N-2=8-2=6
c[2*w+k=2*6+1₀=13]=7 < 8(true)
[7]: '7'
```