Основы теории информации и кодирования

1 сентября 2025 г.

Глава 1. Общие положения данного пособия

Для современных архитектур общего назначения, таких как x84/amd64 или ARM, байт занимает $k = 8$ бит (октет). В программах для них а) символ кодирования — один байт; б) под количественные величины (L для RLE, L и S для LZ77) также отводится около байта (приблизительно k бит):
— один байт — число размером $k = 8$ бит;
— один байт — число + флаг-бит, на число отводится $k - 1 = 7$ бит;
— два байта — пара чисел, каждое — от $k - 2 = 6$ бит до $k + 2 = 10$ бит.
Файлы, для которых имеет смысл отдавать для одной величины два байта и более, встречаются редко (раздел 3.4 и т. п.).

Формулы в настоящем пособии — для k-битного байта в предположении, что:
а) символ кодирования занимает ровно k бит (символ=байт);
б) количественные величины — приблизительно k бит (число≈байт).

Для ЦСП или других архитектур специального назначения байт иногда занимает 4, 12 или 16 бит, причём случай $k > 8$ бит в байте описан стандартом C++. Не стандартизированные языки, подобные C/C++, реализуются и для $k = 4$.

Размер байта k, некратный четырём битам, в настоящее время встречается ещё реже. Тем не менее, исторически использовались байты из $k \in [6, 9]$ бит; весьма распространены были байты из $k = 6$ бит, вмещающие латинский алфавит и цифры (иногда наследием шестибитных программ называют кодировку base64).

При необходимости проиллюстрировать работу алгоритмов сжатия на компактных примерах в настоящем пособии:

а) используется символ из $k = 3$ бит $\big($алфавит $A = \{0, 1, 2, 3, 4, 5, 6, 7\}\big)$ или $k = 4$ бит $\big($алфавит $A = \{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{F}\}\big)$;
б) **байт** считается k-битным, количественные величины занимают около k бит.

Это позволяет продемонстрировать переполнение байта и другие особые случаи на сообщениях, умещающихся в одну-две строки пособия.

Трёхбитные количественные величины L или L и S не позволяют реализовать эффективное сжатие, что видно в том числе и по приводимым примерам. При необходимости на практике реализовать один из методов кодирования для архитектуры с 4- или 16-битным байтом — обычно используется 8-битный символ (составленный из двух 4-битных байтов или полученный битовыми операциями). Это, в частности, облегчает обмен данными с x84/amd64.

Глава 2. Энтропийное кодирование

...А поскольку способности он выказывал необычайные, <...>
имя его обозначалось одной только — да к тому же немой — буквой.

Станислав Лем. Звёздные дневники Ийона Тихого.
Путешествие тринадцатое

Дерево оптимального префиксного кода должно быть сбалансированным, что выражается одновременно в трёх свойствах:

а) длина ветви листа a в рёбрах должна стремиться к $I_{\text{БП}}(a)$ в битах;
б) для каждого родительского узла веса дочерних ветвей должны стремиться к равенству между собой;
в) самые длинные ветви (длинные коды) соответствуют самым редким листам (следует из а)).

Код Шеннона основан на а), Шеннона—Фано — на б), Хаффмана — на в).

Глава 3. Метод кодирования длин повторений (RLE)

> Рыбак сел в лодку и перевёз одну козу, потом вернулся
> и перевёз вторую, потом опять вернулся и перевёз третью...
>
> *Мигель де Сервантес Сааведра. Дон Кихот*

Повторение символа c подряд L раз — цепочку длины $L_{\min} \leqslant L \leqslant L_{\max}$ — будем записывать как пару $\{L, c\}$ — сжатую цепочку:

— повторение одного символа c более чем $L_{\max}$ раз подряд — делится на несколько цепочек длины $L \leqslant L_{\max}$;

— последовательность символов, где ни один не повторяется $L_{\min}$ раз подряд — рассматривается как несжатый текст и кодируется отдельно.

Схемы данных кодирования и декодирования методом RLE представлены на рис. 3.1–3.6 соответственно.

Разделение сжатых цепочек RLE и несжатого текста

Исходный текст $C = c_1 \dots c_n$ в общем случае — произвольная последовательность байтов. Соответственно, любой код пары $\{L, c\}$ теоретически может встретиться в C, следовательно, необходимо отделять код $\{L, c\}$ от кода несжатого текста. Основных способа три:

а) несжатого текста нет ($L_{\min} = 1$) — следуя Мику ван Гельдерену, назовём такую реализацию наивной;

б) несжатый текст разбивается на фрагменты длины $1 \leqslant L \leqslant L_{\max}^{\text{несж}}$:
   — каждый фрагмент несжатого текста предваряется длиной (наподобие строки в стиле Pascal) и маркером несжатого текста;
   — каждая сжатая цепочка предваряется маркером сжатого текста;
   маркер сжатого/несжатого текста (два состояния) — один бит **(флаг-бит),** флаг-бит θ и код длины $\widetilde{L}$ помещаются в один байт;

в) из алфавита выбирается один символ $a = p$ и назначается специальным символом (односимвольным **префиксом):**
   — код каждой сжатой цепочки $\{L, c\}$ предваряется префиксом p (цепочка записывается тремя байтами $p\, b_1\, b_2$);
   — каждый символ $c_i \neq p$ несжатого текста записывается «как есть» (одним байтом c_i);
   — символ $c_i = p$ несжатого текста записывается таким кодом из нескольких байтов, который начинается с p и не совпадает с началом никакого кода $\{L, c\}$ — иначе говоря, символ p в несжатом тексте *экранируется.*

В рамках данного пособия коды, использующие эти способы, будут обозначаться соответственно как RLE-н (наивная реализация), RLE-θ (реализация с флаг-битом сжатая/несжатая цепочка) и RLE-p (с односимвольным префиксом).

3.1. Наивная реализация метода RLE (RLE-н)

Любое количество повторений, включая однократное, рассматривается как сжатая цепочка ($L_{\min} = 1$, рис 3.1–3.2).

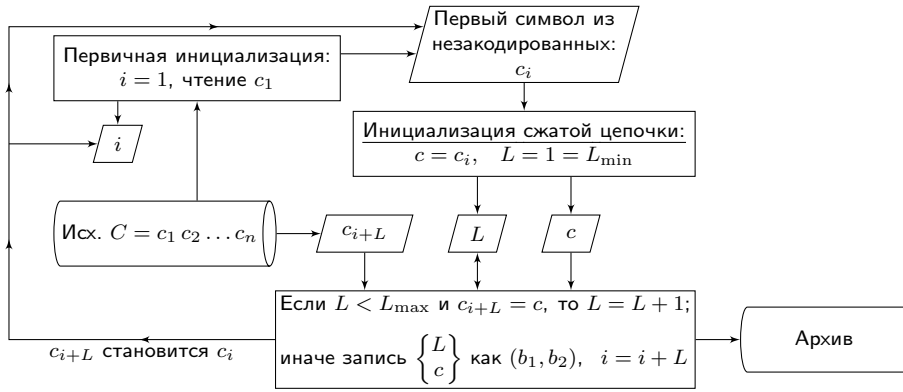

Рис. 3.1. Схема данных кодирования «наивной» реализацией метода RLE

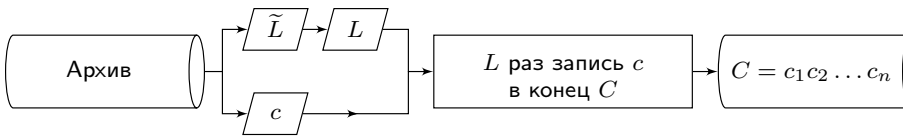

Рис. 3.2. Схема данных декодирования «наивной» реализации метода RLE

Как видно из рис. 3.1, позиция i начала текущей цепочки ни на что не влияет в «наивной» реализации метода RLE, в рассмотренных далее реализациях RLE — также. Переменной i может вообще не существовать — достаточно двух переменных для c_i и c_{i+L}. При чтении c_{i+L} из C не анализируются i и L — это просто следующий непрочитанный символ. На рис. 3.3– 3.5 блок i не показан.

Для кодирования всех файлов используются одни и те же глобальные константы $L_{\min} = 1$ и $L_{\max}$ — определяется исходя из $L_{\min}$ и формата записи L.

3.1.1. Код со смещением для L

Так как несжатого текста в RLE-н не существует, код сжатой цепочки $\{L, c\}$ максимально компактен и состоит только из кода символа c (символа c, запи-

санного байтом «как есть») и кода L (обозначаемого $\widetilde{L}$). Их порядок ($\widetilde{L}c$ или $c\widetilde{L}$) для RLE-н безразличен, так что есть как минимум два варианта «наивной» реализации; далее рассмотрим вариант $\widetilde{L}c$.

Рассматриваем случай, когда код L (обозначим этот код $\widetilde{L}$) занимает один байт, как и c. Случай, когда $\widetilde{L}$ занимает два и более байта, описан в разделе 3.4.

Для k-битного байта $\widetilde{L}$ доступно 2^k кодовых комбинаций и соответствующий диапазон значений:

$$0 \leqslant \widetilde{L} \leqslant 2^k - 1.$$

Запись L без смещения

Простейший способ записи L — это запись байтом «как есть» ($\widetilde{L} = L$). В этом случае из 2^k кодовых комбинаций используется только $2^k - 1$:

$$\begin{cases} 0 \leqslant L \leqslant 2^k - 1 \\ L \geqslant L_{\min} = 1 \end{cases} \implies 1 \leqslant L \leqslant 2^k - 1.$$

Соответственно, для такого формата записи $L_{\max} = 2^k - 1$.

Запись L со смещением

Разработаем такой код, чтобы минимальная возможная кодовая комбинация $\widetilde{L} = 0$ соответствовала минимальному допустимому значению $L = L_{\min} = 1$: будем записывать байт $\widetilde{L} = L - 1$ (код со смещением 1). Соответственно, при декодировании, прочитав $\widetilde{L}$, восстановим $L = \widetilde{L} + 1$.

Тогда используются все кодовые комбинации:

$$\begin{cases} 0 \leqslant L - 1 \leqslant 2^k - 1 \\ L \geqslant L_{\min} = 1 \end{cases} \implies 1 \leqslant L \leqslant 2^k,$$

и $L_{\max} = 2^k$.

Обозначим такой код RLE-н-1 (наивная реализация RLE со смещением 1), а код, где L записывается без смещения — RLE-н-0 (со смещением 0).

Сравнение RLE-н-1 и RLE-н-0

Для трёхбитного байта $L_{\max}^{\text{RLE-н-1}} = 2^3 = 8$ против $L_{\max}^{\text{RLE-н-0}} = 2^3 - 1 = 7$. Сравнение кодов RLE-н-0 и RLE-н-1 в этом случае показано в таблице 3.1.

Независимо от размера байта k:

— код RLE-н-0 более нагляден, поэтому часто рассматривается в иллюстративных целях;

— у кода RLE-н-1 выше степень сжатия: сжатый RLE-н-1 файл всегда не длиннее и иногда короче (в особом случае может быть вдвое короче), чем тот же файл, сжатый RLE-н-0;

— при этом скорость кодирования/декодирования RLE-н-1 и RLE-н-0 одинакова, так как временные затраты на сложение/вычитание чисел пренебрежимо малы по сравнению с ветвлениями и особенно с чтением/записью в файл.

Для восьмибитного байта $L_{\max}^{\text{RLE-н-1}} = 2^8 = 256$, $L_{\max}^{\text{RLE-н-0}} = 2^8 - 1 = 255$.

Сравнение RLE-н-0 и RLE-н-1 для трёхбитного байта

Таблица 3.1

№ п/п	Исходный файл	RLE-н-0 $\left(L_{\max}^{\text{RLE-н-0}} = 7\right)$	RLE-н-1 $\left(L_{\max}^{\text{RLE-н-1}} = 8\right)$
1	1111 2223 3333 2	41 32 53 12	31 22 43 02
2	1	11	01
3	1111 111	71	61
4	1111 1111	71 11	71
5	1111 1111 1	71 21	71 01
6	1111 1111 1111 1111	71 71 21	71 71
7	1111 1111 2222 2222	71 11 72 12	71 72

Случай, аналогичный строке 7 таблицы 3.1 (код RLE-н-1 вдвое короче RLE-н-0) достигается и здесь — в частности, если в файле идёт подряд 256 байтов 01, затем подряд 256 байтов 02 и т. д.

Вообще, если для RLE-кодов A и B $L_{\max}^{A} < L_{\max}^{B}$

3.1.2. Обработка конца файла

При кодировании конец файла при попытке чтения c_{i+L} обрабатывается как $c_{i+L} \neq c$.

3.2. Реализация метода RLE с флаг-битом сжатая/несжатая цепочка (RLE-θ)

Для кодирования всех файлов используется один и тот же набор из четырёх глобальных констант $L_{\min}^{\text{сж}}, L_{\max}^{\text{сж}}, L_{\min}^{\text{несж}} = 1, L_{\max}^{\text{несж}}$:

а) $L_{\min}^{\text{сж}} \in \{2, 3\}$ — выбирается на этапе реализации кодека;

б) $L_{\max}^{\text{сж}}$ определяется исходя из $L_{\min}^{\text{сж}}$ и формата записи L для сжатой цепочки;

в) $L_{\min}^{\text{несж}} = 1$ всегда;

г) $L_{\max}^{\text{несж}}$ определяется исходя из формата записи L для несжатой цепочки.

Если $L_{\min}^{\text{сж}} = 1$ — несжатый текст в принципе отсутствует (вариант наивного RLE), так что RLE-θ имеет смысл рассматривать при $L_{\min}^{\text{сж}} \geqslant 2$. Обычно выбирается $L_{\min}^{\text{сж}} \in \{2, 3\}$, лучшего из них не существует: так, для файла, содержащего последовательность вида 1221 выгоднее $L_{\min}^{\text{сж}} = 3$, а для 1122 — $L_{\min}^{\text{сж}} = 2$.

Конец файла при попытке чтения окончания E (то есть если в конце кодирования получится E короче $L_{\min}$ символов) обрабатывается как ситуация «не все символы E одинаковы»; аналогично для G.

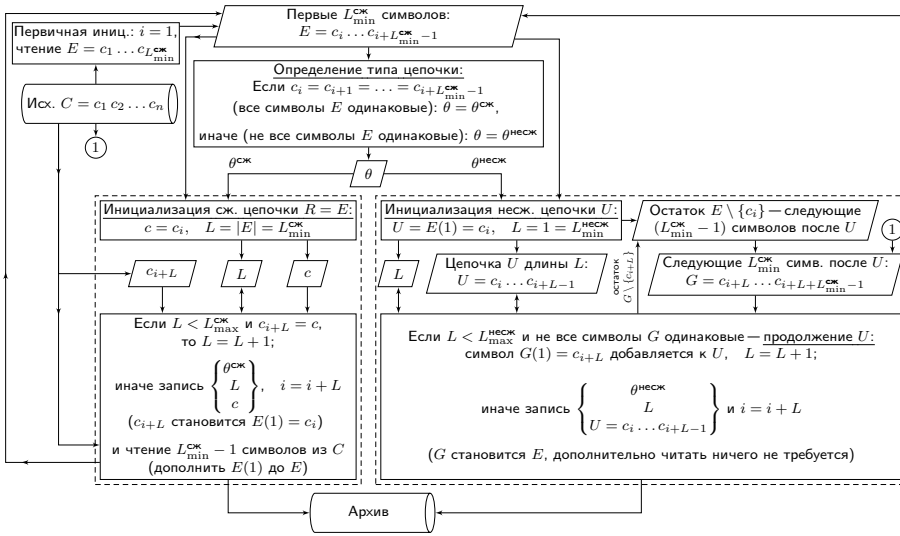

Рис. 3.3. Схема данных кодирования реализацией метода RLE с флаг-битом сжатая/несжатая цепочка

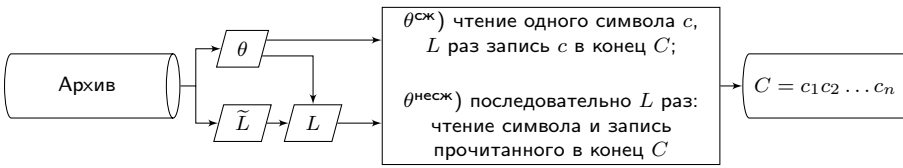

Рис. 3.4. Схема данных декодирования реализации метода RLE с флаг-битом сжатая/несжатая цепочка

3.3. Реализация метода RLE с односимвольным префиксом

Для кодирования всех файлов используется один и тот же набор из четырёх глобальных констант $L^{c\neq p}_{\min} = 4, L^{c\neq p}_{\max}, L^{p}_{\min}, L^{p}_{\max}$:

а) $L^{c\neq p}_{\min} = 4$;

б) $L^{c\neq p}_{\max}$ определяется исходя из $L^{c\neq p}_{\min}$ и формата записи L;

в) $L^{p}_{\min} \in \left[1, L^{c\neq p}_{\min}\right]$ — выбирается на этапе реализации кодека;

г) $L^{p}_{\max}$ определяется исходя из $L^{p}_{\min}$ и формата записи L.

Оптимального $L^{p}_{\min}$ не существует: для каждого $L^{p}_{\min}$ есть файл, который именно при таком $L^{p}_{\min}$ сжимается лучше всего.

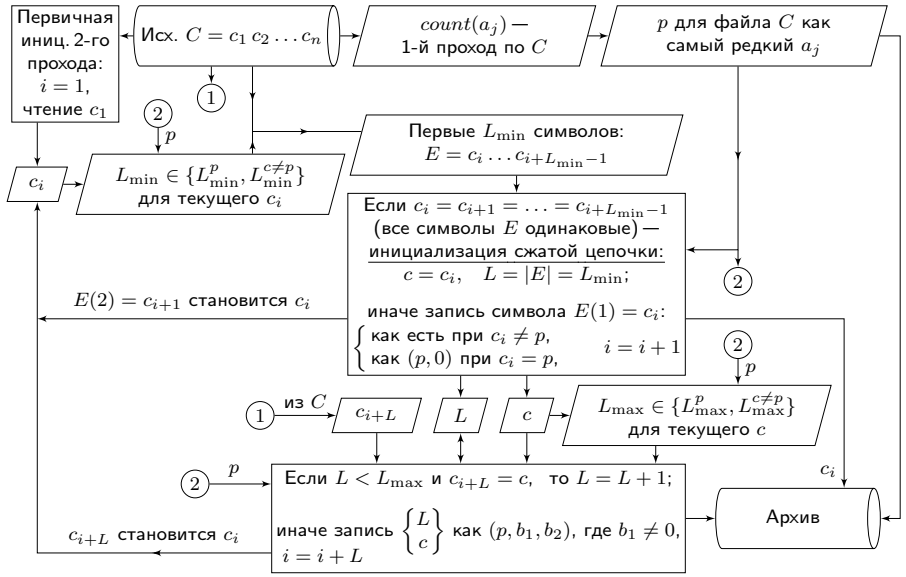

Рис. 3.5. Схема данных кодирования реализацией метода RLE с односимвольным префиксом

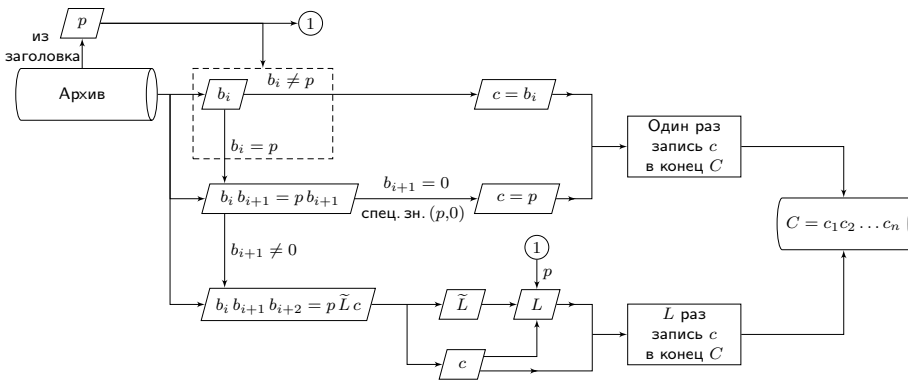

Рис. 3.6. Схема данных декодирования реализации метода RLE с односимвольным префиксом

Для каждого конкретного файла первым проходом по файлу рассчитываются частоты байтов и выбирается префикс p как самый редкий (в идеале — отсут-

ствующий в файле) байт; значение p сохраняется в заголовке архива. Собственно кодирование RLE при заданном p — второй проход.

3.4. Модель источника для «наивного» RLE

Глава 4. Словарные методы сжатия

— Что это вы делаете?
— Рассказываем анекдоты. Но для экономии времени мы их пронумеровали.

Фольклор

4.1. Словарные методы, где словарём является несжатый текст: семейство LZ77

Схемы данных кодирования и декодирования методом LZ77 представлены на рис. 4.2 и 4.2 соответственно.

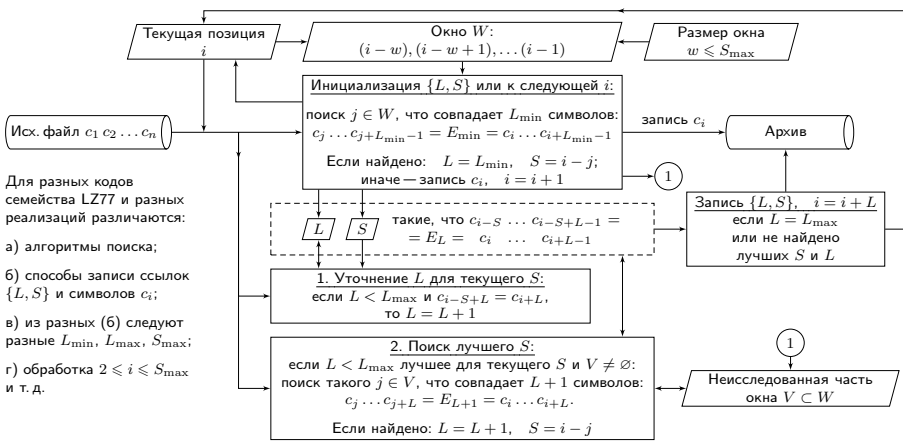

Рис. 4.1. Схема данных кодирования методом LZ77
(любой код семейства LZ77)

Ключевой и самый времязатратный этап любого **алгоритма** кодирования семейства LZ77 — поиск в окне вхождений слова-эталона $E_L = c_i \dots c_{i+L-1}$ длины L (блоки процесса «Инициализация $\{L,S\}$» и «Поиск лучшего S»).

Отметим два момента, которые могут быть неочевидны. Во-первых, слова «вхождение E_L в окне» не подразумевают, что E_L лежит в окне *целиком*. Для LZ77 достаточно, чтобы в окне находилась *начальная позиция* j такая, что $c_j \dots c_{j+L-1}$ составляют E_L.

Во-вторых, кодирование методом LZ77 принципиально неоднозначно: в окне может быть несколько подходящих позиций j, то есть несколько подходящих

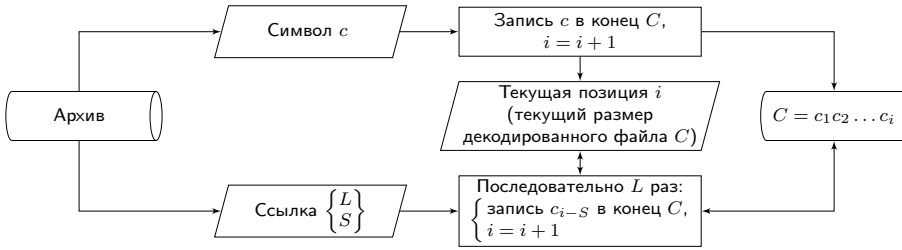

Рис. 4.2. Схема данных декодирования методом LZ77 (декодирование символа c и ссылки $\{L, S\}$ должно строго соответствовать их кодированию)

смещений $S = i - j$, каждому из которых соответствует своя длина L; какое из них будет выбрано — определяется реализацией поиска. Поиск может отличаться направлением исследования окна — по возрастанию позиции (от $i - w$ к $i - 1$), по убыванию (от $i-1$ к $i-w$), с использованием вспомогательных структур данных (первое найденное вхождение может быть не первым и не последним в окне).

При однотипной реализации поиска — **чем быстрее поиск, тем** хуже его результат (**меньше степень сжатия** одного и того же файла). В частности, для ускорения может быть реализован поиск первого подходящего вхождения, а не наилучшего совпадения — тогда блока «Поиск лучшего S» нет, смещение S не меняется от инициализации до записи $\{L, S\}$.

Размер окна w для максимального сжатия должен достигать $S_{\max}$ (все допустимые кодовые комбинации $\widetilde{S}$ должны быть использованы), при этом:

а) первый символ — c_1 — всегда записывается как несжатый символ (окна нет);
б) для $i \geqslant S_{\max} + 1$ доступно окно $W = [i - S_{\max}, i - 1]$ размера $w = S_{\max}$;
в) для $2 \leqslant i \leqslant S_{\max}$ возможны два варианта:
   — поиск в уменьшенном окне $W = [1, i - 1]$ — лучше сжимает некоторые файлы, где в начале много повторений;
   — запись всех c_i из $2 \leqslant i \leqslant S_{\max}$ «как есть», без служебных структур — всегда проще; лучше сжимает те файлы, где повторений в начале мало.

В некоторых реализациях размер окна может быть дополнительно ограничен сверху для ускорения поиска (что ещё сильнее уменьшает степень сжатия).

Ключевая характеристика любого **кода** семейства LZ77 — способы записи ссылок $\{L, S\}$ и несжатых символов c_i. Так как любой код ссылки $\{L, S\}$ теоретически может встретиться в исходном тексте $C = c_1 \dots c_n$, нельзя записывать c_i «как есть» — необходимо отделять код $\{L, S\}$ от кодов сочетаний c_i.

Из разных способов записи ссылок $\{L, S\}$ и несжатых символов c_i следуют разные $L_{\min}$, $L_{\max}$, $S_{\max}$. При этом $S_{\min} = 1$ для любой реализации: технически дополнительно ограничить S снизу возможно, но бессмысленно.

Сочетание **различных кодов семейства LZ77 с различными алгоритмами** поиска порождает неисчислимое количество реализаций идеи LZ77. Такое разнообразие в сочетании с эффективностью сжатия приводит к тому, что многие из этих реализаций известны не как LZ77, а под собственными именами (LZSS от Сторера и Сжимански, LZJB от Джефа Бонвика и др.).

Декодирование любого LZ77 однозначно; скорость декодирования — высокая.

Скорость кодирования LZ77 — низкая (для каждой позиции i необходим поиск в большом окне), что является основным недостатком этого метода и ограничивает его применение. Многие современные реализации LZ77 сжимают *хуже,* чем описанные ранее — зато делают это быстрее.

4.1.1. Алгоритм A1 семейства LZ77, подсемейство «с односимвольным префиксом»

Рассмотрим один алгоритм из огромного семейства LZ77. Так как он хронологически первый из обозреваемых в настоящем пособии, обозначим его A1.

Как всегда, **символ=байт.** Перечислим опции алгоритма и кода A1.

1. Для отделения ссылок $\{L, S\}$ от несжатого текста используется односимвольный префикс (см. раздел 4.2).
2. Как L, так и S занимают по одному байту, k бит (напомним, что в этой главе $k = 4$ — но принципиальных отличий реализации от $k = 8$ нет).
3. Вначале записывается байт L, затем байт S.
4. Для значений L и S используется код без смещения ($\Delta_L = \Delta_S = 0$). Это **проигрышный выбор для любого кода** с точки зрения объёма и не увеличивает скорость. Для иллюстрации метода LZ77 он выбран только из-за наглядности полученного кода для человека.
5. Если есть несколько байтов, в равной степени подходящих на роль префикса — из них выбирается наименьший по значению.
6. Начинаем поиск совпадений с позиции $i = w+1$, где доступно полноразмерное окно — первые w символов записываем «как есть».
7. Если после окончания поиска совпадений среди последних символов встречается префикс, он экранируется.
8. Поиск в окне — последовательный по возрастанию позиции (от $i - w$ к $i - 1$).
9. Ищем наилучшее вхождение. Это выигрышный выбор с точки зрения объёма, но крайне проигрышный по скорости.
10. Заканчиваем поиск совпадений на позиции $i = n - L_{\max} + 1$, где ещё доступен полноразмерный эталон.
11. Не ограничиваем размер окна: $w = S_{\max}$. По сравнению с $w < S_{\max}$ — выигрыш по объёму, проигрыш по скорости.

Опции 1–11 определяют алгоритм кодирования и сам код. Из них:

— опции 1–7 влияют также и на алгоритм декодирования;

— если изменить опции 8–11 — алгоритм кодирования и код изменятся, но всё ещё будет применим декодер A1.

Все опции, кроме 5, имеют смысл для любого способа отделения ссылок $\{L, S\}$ от несжатого текста; способ выбора префикса 5, очевидно, актуален только в том случае, если для отделения используется односимвольный префикс.

Следствия из выбранных опций:

1. Из 1 (ссылка с учётом префикса записывается *тремя* байтами) следует $L_{\min} = 4$.
2. Из 4 и 2 (оба L и S записываются без смещения k битами) $L_{\max} = S_{\max} = 2^k - 1$. С учётом $k = 4$ в этой главе $L_{\max} = S_{\max} = 2^4 - 1 = 15_{10} = \texttt{F}$.
3. Из 11 и $S_{\max} = 15_{10}$ размер окна также $w = S_{\max} = 15_{10} = \texttt{F}$ позиций.
4. Из 6 (первые w символов записываем «как есть») следует, что и префикс среди первых $w = 15_{10} = \texttt{F}$ символов экранировать не нужно.

Кодирование A1

Рассмотрим сообщение C вида

`012340123456789012345678901234540123ABCDEFFFFFFF0123EFCD456789AB`

из $n = 64_{10} = 80_8$ байтов. Его код A1 имеет вид

$$A1(C) = \texttt{0123401234567896FA54647ABCDEF66168F4560789AB}$$

и состоит из $44_{10} = 54_8$ байтов (рис. 4.3).

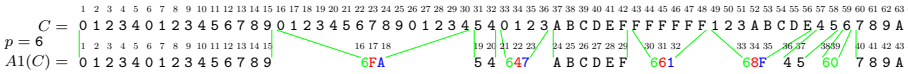

Рис. 4.3. Сообщение C и его код A1

Рассмотрим процесс его построения подробно. Так как, согласно опции 1, ссылка в коде A1 отделяется от несжатого текста префиксом, для сообщения C в первую очередь необходимо выбрать байт p, который будет префиксом. Для этого необходимо выполнить предварительный проход по файлу (соответственно, код A1 не может быть поточным кодом) и найти самый редкий символ кодируемой части файла.

Так как, согласно опции 6, первые $15_{10} = \texttt{F}$ символов записываются «как есть», префикс экранируется только начиная с позиции 16_{10}. Необходимо учитывать статистику только этой части (рис. 4.4).

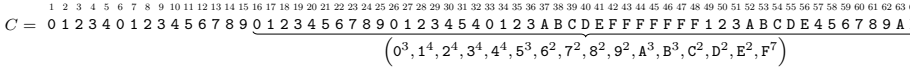

Рис. 4.4. Выбор префикса для сообщения C (алгоритм A1)

Префиксом для сообщения C может быть любой из одинаково редких байтов `6`, `7`, `8`, `9`, `C`, `D`, `E`. Согласно опции 5, выбираем из них байт `6`:

$$p = \texttt{6}. \tag{4.1}$$

Значение `6` записывается в соответствующее поле заголовка.

Начало кода $A1(C)$ — первые $15_{10} = \texttt{F}$ символов сообщения C «как есть»:

$$A1(C) = \texttt{012340123456789...},$$

далее, начиная с позиции $i = 16$, может быть записана либо ссылка в виде тройки байтов $(\texttt{6}, L \geqslant 4, S \geqslant 1)$, либо байт несжатого текста (`6` экранируется). Для того, чтобы определить, что именно — необходимо рассмотреть окно (рис. 4.5) и для

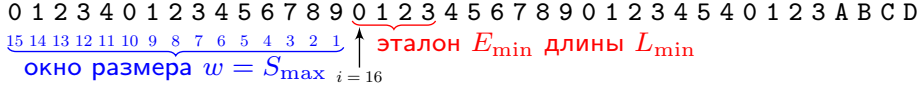

Рис. 4.5. Окно и эталон для позиции $i = 16$

каждой его позиции сравнить строку на этой позиции с эталоном $E_{\min}$.

Для каждой позиции окна внизу рис. 4.5 показано соответствующее смещение S относительно текущей позиции кодирования i. На этом и последующих рисунках показана только часть позиций сообщения C, чтобы изображение не было слишком мелким.

Согласно опции 8, окно просматривается по возрастанию позиции, то есть слева направо — от $S = S_{\max}$ до $S = 1$. На рис. 4.6, а–в) показан анализ первой позиции окна: $S = S_{\max} = 15_{10}$. Из рис. 4.6, а) видно, что четырёхбайтовое слово

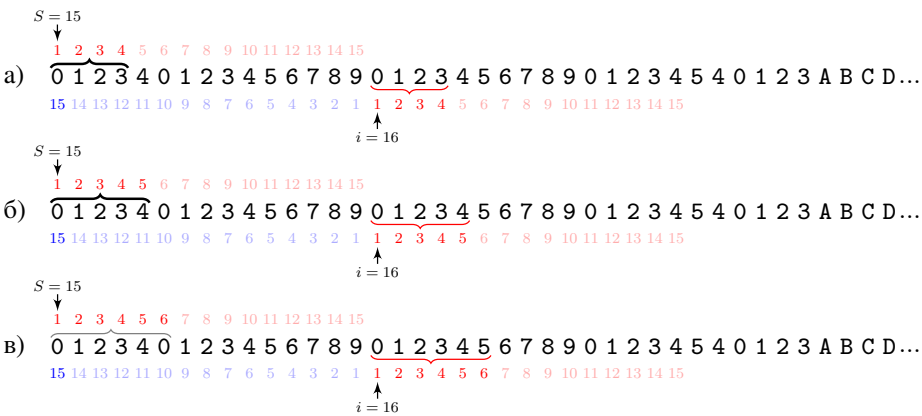

Рис. 4.6. Кодирование C, $i = 16$, смещение $S = 15_{10}$

по смещению $S = 15_{10}$ совпадает с четырёхбайтовым эталоном $E_{\min} = \texttt{0123}$ на позиции i. Следовательно, в коде далее будет ссылка на слово длины $L \geqslant 4$.

Рис. 4.6, б)–в) показывают *уточнение L для текущего S:* видно, что не только четырёх-, но и пятибайтовые слова по смещению $S = 15_{10}$ и на позиции i совпадают, а вот шестибайтовые уже различаются. Таким образом, ссылку можно сохранить на слово длины $L \geqslant 5$, и одна подходящая ссылка уже найдена: это $\{L = 5, S = 15_{10}\}$. Если бы опция 9 предписывала поиск не *наилучшего,* а *первого подходящего* вхождения, то $\{L = 5, S = 15_{10}\}$ и была бы записана в файл.

Но опция 9 требует искать наилучшее, поэтому продолжаем просматривать остаток окна (от смещения $S = 14_{10}$ до $S = 1$) и ищем лучшее, чем пять байтов, совпадение, для чего сравниваем уже шестибайтовые слова с шестибайтовым эталоном $E_6 = \texttt{012345}$ (рис. 4.7, а–е)

Рис. 4.7, а–д) показывают *поиск лучшего S:* по смещениям от $S = 14_{10}$ до $S = 11_{10}$ (а–г) нет искомого $E_6 = \texttt{012345}$, а по смещению $S = 10_{10}$ (д) есть.

После обнаружения лучшего S для него снова *уточняется* L (рис 4.7, е). Для $S = 10_{10}$ совпадают уже $L = 15_{10} = L_{\max}$ байтов. Так как достигнуто значение $L_{\max}$, лучшего совпадения уже не найти — просмотр окна останавливается, ссылка $\{L = 15_{10} = \texttt{F}, S = 10_{10} = \texttt{A}\}$ записывается тройкой байтов $(\texttt{6}, \texttt{F}, \texttt{A})$:

$$A1(C) = \texttt{0123401234567896FA}\ldots$$

Этой ссылкой были записаны пятнадцать байтов сообщения C. Таким образом, следующая позиция кодирования — $i = 16 + 15 = 31$ (рис. 4.8, а–б). Как всегда при переходе к новой i, производится поиск минимально допустимого четырёхбайтового эталона $E_{\min} = \texttt{5401}$. Последовательное сравнение со всеми смещениями в окне (между а и б на рис. 4.8 пропущено ещё тринадцать шагов) показывает, что ни по одному смещению нет совпадения с $E_{\min}$. Следовательно, следующей записью будет не ссылка, а один несжатый байт 5:

$$A1(C) = \texttt{0123401234567896FA5}\ldots$$

Следующая позиция кодирования — $i = 32$ (рис. 4.9). Аналогично, пятнадцать сравнений (на рис. 4.9 изображено только первое, с $S = 15_{10}$) показывают, что в окне нет ни одного совпадения даже с $E_{\min} = \texttt{4012}$, так что следующей записью будет один байт 4:

$$A1(C) = \texttt{0123401234567896FA54}\ldots$$

Следующая позиция кодирования — $i = 33$ (рис. 4.10, а–д). По первому смещению $S = 15_{10}$ (рис. 4.10, а) совпадения с $E_{\min} = \texttt{0123}$ нет, но через семь шагов на $S = 7$ (рис. 4.10, б) совпадение находится. Попытка уточнения L (рис. 4.10, в) показала, что пятибайтовые слова уже не совпадают: найдено $\{L = \texttt{4}, S = \texttt{7}\}$.

Так как опция 9 требует искать наилучшее совпадение, ссылка $\{L = \texttt{4}, S = \texttt{7}\}$ не может быть записана до тех пор, пока остаток окна не просмотрен до конца и не доказано, что нет совпадения с $E_5 = \texttt{0123A}$ (рис. 4.10, г–д).

После окончания просмотра окна ссылка записывается:

$$A1(C) = \texttt{0123401234567896FA54647}\ldots$$

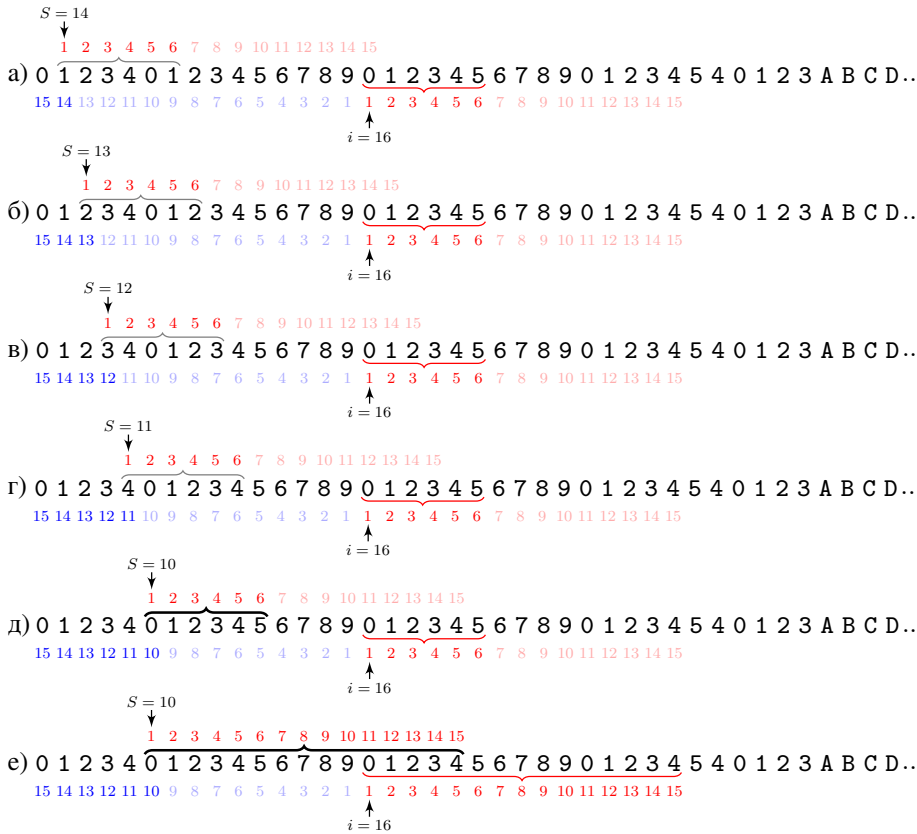

Рис. 4.7. Кодирование C, $i = 16$,
смещения от $S = 14_{10}$ до $S = 10_{10}$

Таким образом, в общем случае необходимо на *каждую* позицию i исходного сообщения (чтобы в итоге записать в выходной поток один символ или ссылку) проанализировать *все* $w = S_{\max}$ позиций окна, причём для каждой позиции сравнить как минимум четыре символа. Поэтому все те варианты LZ77, где выполняется поиск *наилучшего совпадения* — исключительно медленны.

Следующая позиция кодирования — $i = 37$ (рис. 4.11). В окне нет ни одного совпадения с эталоном `ABCD`, следующей записью будет один байт `A`:

$$A1(C) = \texttt{0123401234567896FA54647A...}$$

Аналогично следующие пять записей также будут байтами:

$$A1(C) = \texttt{0123401234567896FA54647ABCDEF...}$$

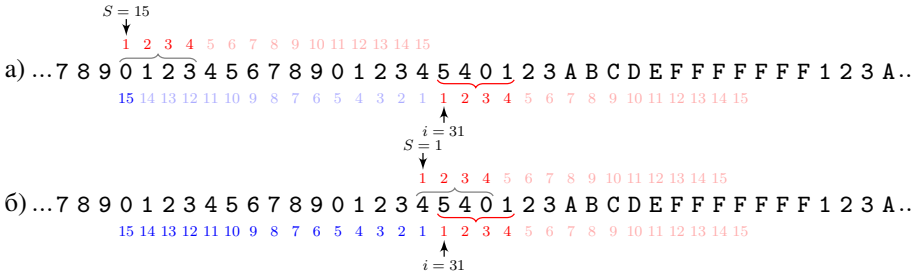

Рис. 4.8. Кодирование C, $i = 31$

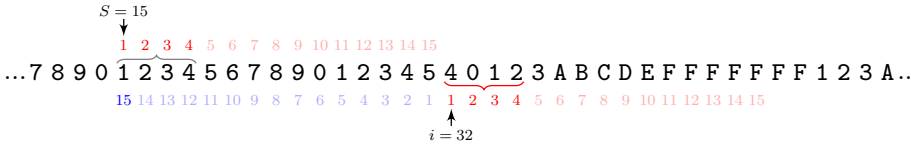

Рис. 4.9. Кодирование C, $i = 31$

На позиции $i = 43$ (рис. 4.12, а–в) совпадение найдено только на последнем шаге $S = 1$ (рис. 4.12, в), после уточнения L получим $\{L = 6, S = 1\}$:

$$A1(C) = 0123401234567896FA54647ABCDEF661...$$

На следующей позиции $i = 43$ (рис. 4.13, а–б) также находим совпадение $\{L = 8, S = F\}$ и после того, как убедимся, что далее в окне нет ещё лучшего совпадения, записываем ссылку:

$$A1(C) = 0123401234567896FA54647ABCDEF66168F...$$

Следующая позиция кодирования $i = 57$ (рис. 4.14) превышает заданную в опции 10 границу $i_{\max} = n - L_{\max} + 1 = 64 - 15 + 1 = 50$ — останов поиска совпадений.

Отметим, что если бы поиск совпадений продолжался — его алгоритм бы отличался от описанного ранее, так как в позиции кодирования $i = 57$ (рис. 4.14) недоступен эталон максимальной длины $L_{\max} = 15$, а только до конца файла (восемь символов).

После останова поиска совпадений и до конца файла записываться будут несжатые байты, но префикс 6 по опции 7 экранируется:

$$A1(C) = 0123401234567896FA54647ABCDEF66168F4560789AB$$

Отметим, что в кодируемой части C символ 6 встречается *два* раза, а экранировать его пришлось только *один* раз — первое вхождение стало частью ссылки. Такое совпадение удачно сократило объём кода, но предсказать его заранее по частотам символов было невозможно.

Таким образом, у кодера A1 есть три интервала с разными режимами работы:

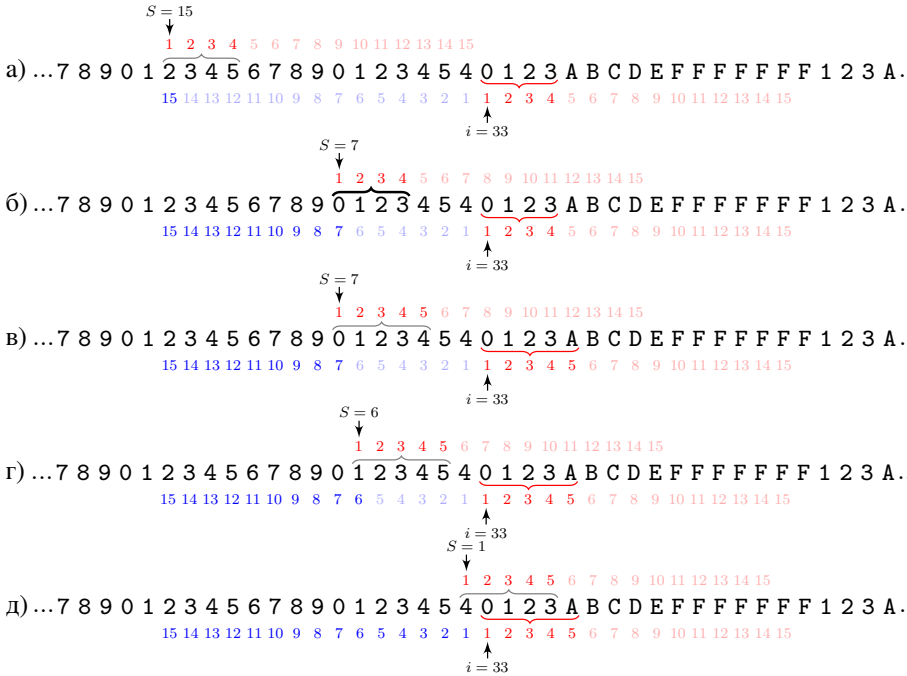

Рис. 4.10. Кодирование C, $i = 33$

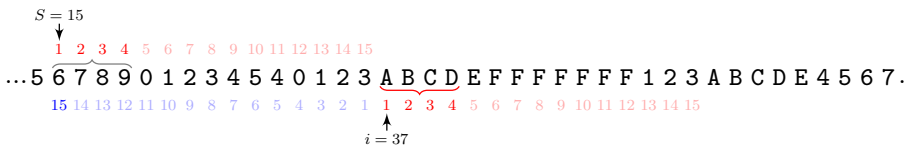

Рис. 4.11. Кодирование C, $i = 37$

а) $1 \leqslant i \leqslant S_{\max}$ — символы переписываются «как есть»: без ссылок, префикс не экранируется;

б) $S_{\max} + 1 \leqslant i \leqslant n - L_{\max} + 1$ — «нормальное» кодирование:
   — окно длины $w = S_{\max}$;
   — эталон длины от $L_{\min}$ (в начале поиска) до $L_{\max}$;
   — префикс экранируется;

в) $i \geqslant n - L_{\max} + 2$ — каждый очередной символ записывается как несжатый, префикс экранируется.

У декодера A1 будет только два интервала с разными режимами работы:

Рис. 4.12. Кодирование C, $i = 43$

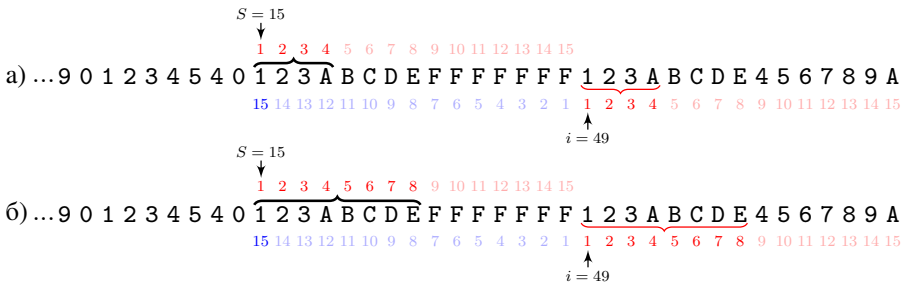

Рис. 4.13. Кодирование C, $i = 49$

Рис. 4.14. Останов поиска совпадений C, $i = 57$

а) $1 \leqslant i \leqslant S_{\max}$ — символы переписываются «как есть»: ссылок быть не может, префикс не анализируется;

б) $S_{\max} + 1 \leqslant i \leqslant n$ — «нормальное» декодирование: анализируется префикс.

Именно ради упрощения декодирования (чтобы «хвост» кода анализировался по общему правилу) в A1 была включена опция 7. Если при кодировании в интервале (в) не экранировать префикс (это будет уже не алгоритм A1, так как опция 7 отличается) — код будет немного короче, но декодирование — сложнее.

Декодирование A1

При декодировании из заголовка файла прочитаны номер алгоритма (A1) и значение префикса ($p = 6$); кроме того, в архиве записан код:

$$A1(C) = 0123401234567896FA54647ABCDEF66168F4560789AB.$$

Известно, что первые 15_{10} байтов A1 записывал «как есть», так что при декодировании они копируются без дополнительного анализа:

$$C = 012340123456789...$$

Счётчик позиции i для декодера увеличивается при каждой записи символа в выходной (уже декодированный) поток, то есть здесь $i = 15_{10}$.

Начиная с позиции $i = 16_{10}$, требуется проверять каждый прочитанный байт на равенство префиксу 6. Так, байт на позиции 16_{10} равен 6 — читаем следующий и сравниваем его с нулём. Следующий $\texttt{F} \neq 0$ — значит, это начало ссылки: $L = \texttt{F} = 15_{10}$, а далее записано $S = \texttt{A} = 10_{10}$.

Таким образом, необходимо взять файл с **уже декодированным** текстом C, прочитать десятый с конца символ и вставить его в конец файла:

$$C = 0123401234567890...,$$

затем ещё раз прочитать десятый с конца символ (конец файла сместился, так что это будет уже другой символ — следующий за прочитанным ранее) и вставить вновь прочитанный символ в конец:

$$C = 01234012345678901...,$$

и повторить это в общей сложности пятнадцать раз:

$$C = 012340123456789012345678901234...$$

Каждый раз символ для копирования существует.

При каждой записи увеличивается i, в итоге получим $i = i + L = 15 + 15 = 30$.

После окончания разбора ссылки 6FA ($i = 30$) читаем следующий байт входного потока — это 5. Он не равен префиксу $p = 6$, поэтому его необходимо переписать в выходной поток «как есть»:

$$C = 0123401234567890123456789012345...$$

Далее действуем аналогично: символ $4 \neq p$ переписываем «как есть»:

$$C = 01234012345678901234567890123454...$$

символ $6 = p$ — префикс: после него ненулевой байт 4, так что это начало ссылки 647 — четыре раза читаем седьмой с конца символ, после каждого раза записываем прочитанное в конец и только после записи можем читать снова:

$$C = 012340123456789012345678901234540123...,$$

далее символы ABCDEF:

$$C = 012340123456789012345678901234540123ABCDEF...$$

и ссылка 661 — шесть раз читаем последний символ файла, после каждого раза записываем прочитанное в конец:

$$C = 012340123456789012345678901234540123ABCDEFFFFFFF...;$$

хотя символы получаются одинаковыми, но источник для копирования каждый раз новый. Случай $S = 1$ не является особым и обрабатывается по общему правилу.

Далее — ссылка 68F:

$C =$ 0123401234567890123456789012345401 23ABCDEFFFFFFF123ABCDE...

и символы 45:

$C =$ 0123401234567890123456789012345401 23ABCDEFFFFFFF123ABCDE45...

Далее символ 6 — префикс: после него 0 — не начало ссылки, а собственно 6:

$C =$ 0123401234567890123456789012345401 23ABCDEFFFFFFF0123EFCD456...,

далее — символы 789AB. Полученное сообщение

0123401234567890123456789012345401 23ABCDEFFFFFFF123ABCDE456789AB

совпадает с исходным по содержанию и длине.

Декодирование любого варианта LZ77 выполняется намного быстрее кодирования: не нужен трудоёмкий поиск, отсчитываем заданное в архиве смещение.

A1 и исходная длина файла n

Рассмотренная здесь реализация A1 метода LZ77 — один из немногих алгоритмов сжатия, которые никогда не дописывают незначащие символы в конце, и, соответственно, при декодировании не требуют исходной длины n. Тем не менее, для дополнительного контроля целостности n лучше сохранить в архиве, а если A1 применяется совместно с каким-либо кодом сжатия без контекста, защиты от помех или шифрования — n необходима для корректного декодирования.

4.1.2. Алгоритм A2 семейства LZ77, подсемейство «с односимвольным префиксом»

Рассмотрим алгоритм **A2** семейства LZ77, который отличается от A1 только опцией 6: **поиск совпадений начинается так рано, как это возможно.** При этом один первый символ всё равно записываются «как есть», так как перед ним нет ни одной позиции. Таким образом, ссылка на предыдущее вхождение невозможна, а значит — если даже первый символ сообщения совпадает с выбранным префиксом, экранировать его нет смысла.

Кодирование начинается только с позиции $i = 2$. Необходимо учитывать статистику от $i = 2$ до конца $i = n$. По этим частотам (рис. 4.15) префиксом, согласно

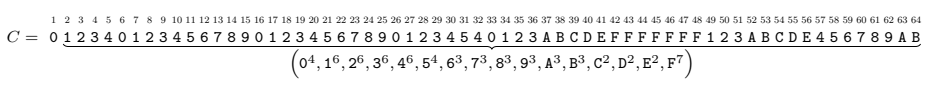

Рис. 4.15. Выбор префикса для сообщения C (алгоритм A2)

опции 5, выбирается уже байт C.

Для позиции $i = 2$ (рис. 4.16) окно состоит из единственного возможного смещения $S = 1$ (позиция $i = 1$). Здесь четырёхбайтовое слово 0123 по смещению $S = 1$ не совпадает с минимально возможным четырёхбайтовым эталоном $E_{\min} =$ 1234, поэтому второй байт 1 записывается как несжатый символ.

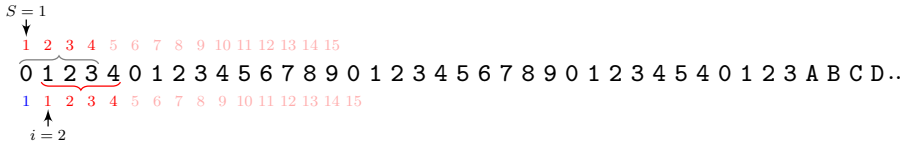

Рис. 4.16. Кодирование C, $i = 2$

Отметим, что для иного сообщения, в частности, для $C = 00000\ldots$, слово по смещению $S = 1$ могло бы совпасть с эталоном и для $i = 2$ могла быть записана ссылка с $S = 1$ и некоторым $4 \leqslant L \leqslant L_{\max}$. Так, для $C_{л16} = 0000000000000000$ из шестнадцати нулевых байтов код $A2(C_{л16}) = 01F1$ (префиксом согласно опции 5 выбран байт 1).

Соответственно, если второй, третий и т. д. символы сообщения совпадают с выбранным префиксом — их в A2 необходимо экранировать.

Для позиции $i = 3$ (рис. 4.17, а–б) окно состоит из двух возможных смещений

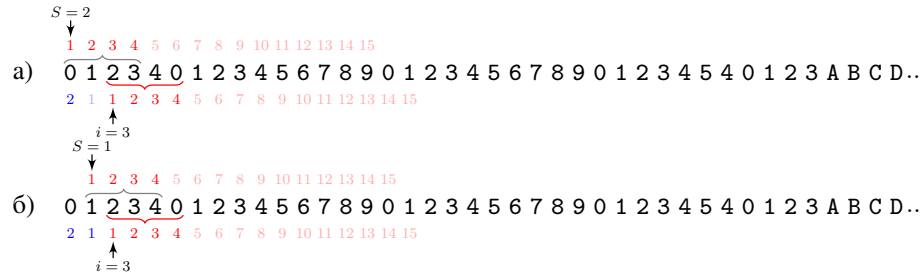

Рис. 4.17. Кодирование C, $i = 3$

$S = 2$ и $S = 1$ (позиции $i = 1$ и $i = 2$), и далее аналогично.

Таким образом, для каждой позиции $2 \leqslant i \leqslant S_{\max} + 1$ окно состоит из $i - 1$ возможных смещений (ограничением является начало файла):

— смещение $S = i - 1$ соответствует позиции 1 сообщения (первой);

— смещение $S = 1$ соответствует позиции $i - 1$ сообщения (непосредственно предшествующей кодируемому слову/символу);

при $i = S_{\max} + 1$ (для A2 при $i = 16$) размер $i - 1$ этого окна достигает $S_{\max}$. Далее кодирование A2 аналогично A1 (рис. 4.18).

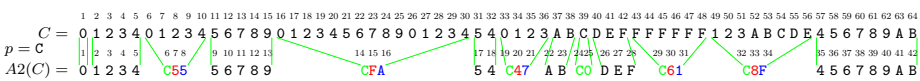

Рис. 4.18. Сообщение C и его код A2

Для рассматриваемого сообщения C код A2 (42 байта) получился на два байта короче, чем A1 (44 байта). Для другого сообщения код A2 может, наоборот, оказаться длиннее A1: если в позициях $2 \leqslant i \leqslant S_{\max}$ нет или мало совпадений, зато часто встречается символ-префикс.

Различие длин кодов A1 и A2 всегда относительно невелико: результат их работы отличается только в первых $S_{\max}$ позициях файла, а любой алгоритм сжатия имеет смысл применять только к достаточно длинным файлам, то есть $n \gg S_{\max}$.

Реализация A2 сложнее A1. У кодера A2 есть **четыре** интервала с разными режимами работы (у кодера A1 — три):

а) $i = 1$ — первый символ переписывается «как есть»: префикс не экранируется;

б) $2 \leqslant i \leqslant S_{\max}$ — кодирование начала файла (используется укороченное окно):
   — окно длины i;
   — эталон длины от $L_{\min}$ (в начале поиска) до $L_{\max}$;
   — префикс экранируется;

в) $S_{\max} + 1 \leqslant i \leqslant n - L_{\max} + 1$ — «нормальное» кодирование:
   — окно длины $w = S_{\max}$;
   — эталон длины $L_{\min}$ (в начале поиска) до $L_{\max}$;
   — префикс экранируется;

г) $i \geqslant n - L_{\max} + 2$ — каждый очередной символ записывается как несжатый, префикс экранируется.

У декодера A2, как и у A1 — два интервала с разными режимами работы:

а) $i = 1$ — первый символ переписывается «как есть»: ссылки быть не может, префикс не анализируется;

б) $2 \leqslant i \leqslant n$ — «нормальное» декодирование: анализируется префикс.

Режимы аналогичны декодеру A1, а интервалы разные: для A2 нужен отдельный не только кодер, но и декодер.

4.1.3. Алгоритм A3 семейства LZ77, подсемейство «с односимвольным префиксом»

Рассмотрим также алгоритм **A3** семейства LZ77, который отличается от A1 только опцией 10: **поиск совпадений заканчивается так поздно, как это возможно.**

4.2. Коды семейства LZ77 с префиксом

4.3. Словарные методы с отдельным словарём: семейство LZ78

Приложение А. Регламент курса

В данном разделе даны базовые положения регламента курса ОТИК. **Все особые случаи обсуждаются с преподавателем индивидуально,** решение в каждом случае принимается отдельно и только **после воплощения соответствующего случая в реальность.** То, что было возможно в начале семестра — недопустимо в конце.

Хотя декабрь загружен не так сильно, как май, но всё же гораздо сильнее октября.

А.1. Итоговая оценка Q

Оценка («неуд»/«удовл»/«хор»/«отл») выставляется по совокупности:
— выполненной в семестре работы;
— обязательного для всех экзамена (или зачёта у НБ-3*).
Баллы в ОРИОКС подгоняются под полученную оценку за счёт граф «Итог» и «Натяжка»; при этом **честно заработанные в семестре баллы не уменьшаются.** Баллы, полученные жульничеством (плагиат, манипуляции с размером команды и т. д. — полный список не приводится, так как нельзя объять необъятное), **удаляются** как техническая ошибка — независимо от того, сколько времени они пробыли в ОРИОКС.

А.1.1. ОРИОКС: сумма баллов S и формирование итоговой оценки Q

Текущая оценка Q записывается в ОРИОКС в виде суммы баллов S.

Шкала оценок

Сумма баллов S и оценка Q связаны соотношением:

$$Q(S) = \begin{cases} \text{«отл»}, & S \geqslant 86, \\ \text{«хор»}, & 70 \leqslant S < 86, \\ \text{«удовл»}, & 50 \leqslant S < 70 \text{ и графа «Натяжка» пуста/0/н}, \\ \text{«3 с натяжкой»}, & S \approx 50 \quad \text{и графа «Натяжка» непуста}, \\ \text{«неуд»}, & S < 50. \end{cases} \tag{А.1}$$

В итоговой ведомости «3 с натяжкой» отображается как «удовл». Внутри семестра считается «неуд» $<$ «3 с натяжкой» $<$ «удовл», и оценка «3 с натяжкой» не может быть повышена до «хор».

КМ, баллы и оценки

Непосредственно на баллы s (линейный рост $S_{\text{новое}} = S_{\text{старое}} + s$) оцениваются:
— лабораторные работы, защищённые на 1–16 неделях по графику (раздел РЛ);
— задачи, решаемые на лекциях (раздел А.2.1);

— работа на семинарах (раздел А.2.1);
— замечания и дополнения (раздел А.6);
На оценку Q:
— все альтернативные лабораторным работам варианты (раздел А.5);
— экзамен/зачёт (раздел А.3);
баллы s при этом добавляются так, чтобы оценка по новой сумме баллов $S_{\text{старое}} + s$ равнялась Q (нелинейный расчёт s не только по оценке Q, но и по $S_{\text{старое}}$).

А.1.2. Графы КМ в ОРИОКС

Так как для получения оценки по ОТИК есть несколько альтернативных вариантов, иногда взаимоисключающих — многие из граф КМ в ОРИОКС могут быть пустыми даже у студента с итоговой $Q =$ «отл». Графа «Натяжка» у нормально успевающего студента *должна* быть пустой.

Баллы и специальные значения граф

В графу КМ ОРИОКС могут быть выставлены значения:
а) $s \geqslant 1$: КМ успешно защищено с баллами s;
б) $s = 0$, $s =$ н или $s = 0{,}01$: защиты обязательного КМ не было;
в) $s = 0{,}02$: маркер «защита провалена» — оценка за КМ «неуд».
КМ с 0/н/$0{,}01$ можно сдать любому преподавателю ОТИК, кто согласится принимать, и повысить оценку; $0{,}02$ — либо тому, кто выставил $0{,}02$, либо комиссии.

Графы лабораторных работ и КР — значение выставляется один раз

В графы лабораторных работ Лi, Кр1 и Кр2 значение s выставляется после защиты (при повышении — заменяется на новое).

Для работы Лi, защищённой на 1–16 неделях по графику:
а) базовые баллы $s \geqslant 1$ выставляются в графу «Лi»;
б) бонусные баллы $s^{\text{bonus}} \geqslant 0$ для Л1–Л3 добавляются в графу «Бонус Л1–Л3», для Л4 и Л5 — к графам соответствующих лабораторных работ.

Накопительные графы бонусов — значение растёт от 0 в течение семестра

Со значениями граф «Зд (лек)», «Бонус (прочее)», «Бонус (л/р)» и т. п. вновь заработанные баллы суммируются ($s_{\text{накоп. новое}} = s_{\text{накоп. старое}} + s$):
а) зд (лек) — задачи, решаемые на лекциях, выставляет лектор;
б) бонус (прочее) — иные баллы, которые назначает лектор и для которых нет отдельной графы (в частности, за замеченные опечатки и ошибки);
в) бонус (л/р):
— бонусные задания всех лабораторных работ Лi;
— иные бонусные баллы, которыми преподаватель отмечает какую-либо Лi.
— а также все прочие баллы, которые назначает преподаватель лабораторных работ (на своё усмотрение) и для которых нет отдельной графы.

Графы семинаров — два раза в семестр

Графа «Зд (сем) <недели>» — баллы за работу на семинарах на соответствующих неделях, без учёта посещаемости; графа «П (сем) <недели>» — отдельно посещаемость семинаров. Значения s выставляются после окончания интервала:

а) за 1–8 недели — на 9 неделе, $0 \leqslant s \leqslant \max(s)$, где $\max(s)$ — ёмкость графы;

б) за 9–16 недели — дважды, предварительные (9–14) и полные (9–16) значения:
   — на 15 неделе — предварительные $0 \leqslant s \leqslant \frac{3}{4}\max(s)$;
   — на 17 неделе — полные $0 \leqslant s \leqslant \max(s)$.

Баллы за каждый интервал (1–8 или 9–16 недели) выставляются независимо друг от друга. Оценивается интервал в целом, «баллы за один плюсик» не определены.

К баллам посещаемости: если студентов мало и я могу опросить всех, то «пришёл просто так посидеть» оценивается как «не был».

Графа «Итог» — значение подгоняет общую сумму баллов S под оценку $Q_{\text{Итог}}$

После успешного блиц-экзамена/зачёта в графу «Итог» выставляется 1 балл.

Итог полного режима экзамена/зачёта — *не фиксированное число баллов, а оценка* за весь курс: «неуд» $\leqslant Q_{\text{Итог}} \leqslant$ «отл».

Если при сумме баллов $S_{\text{старое}}$ до экзамена и оценке $Q_{\text{Итог}}$ на экзамене:

1. $Q_{\text{Итог}} \geqslant Q(S_{\text{старое}})$ — студенту добавляется такое количество s баллов, чтобы сформировалась оценка $Q_{\text{Итог}}$, то есть чтобы $Q(S_{\text{старое}} + s) = Q_{\text{Итог}}$.
   Баллы s распределяются в графы «Итог» и «Натяжка».
2. $Q_{\text{Итог}} < Q(S_{\text{старое}})$ — в зависимости от ситуации возможны варианты:
   — на первой и потенциально единственной попытке сдачи экзамена — в «Итог» выставляется $s = 0{,}02$ балла; прочие баллы (сумма $S_{\text{старое}}$) сохраняются;
   — если после нескольких пересдач оценка на экзамене стабильно хуже оценки по текущим баллам $S_{\text{старое}}$, но при этом оценка по $S_{\text{старое}}$ не устраивает студента — возможен поиск ошибки в старых баллах ОРИОКС и удаление ошибочно выставленного.

Графа «Натяжка»

Если оценка $Q_{\text{Итог}} =$ «3 с натяжкой», то дополняющие до 50 баллы выставляются в графу «Натяжка»: они не соответствуют никаким реальным достижениям студента. Если студент повышает $Q_{\text{Итог}}$ до «удовл» и выше — выставляются баллы за сделанное, а графа «Натяжка» обнуляется.

А.2. Работа в семестре (1–16 недели)

Напомню тот очевидный факт, что работа в семестре включает не только *контроль* знаний, умений и навыков, но и их *получение*. Это невозможно без

обратной связи, поэтому, если что-то непонятно — **спрашивайте!!!** Даже перед экзаменом всегда есть консультация для разрешения вопросов по курсу.

Студент, приступающий к изучению курса ОТИК, по определению ещё его не знает (знающие — могут доказать это в начале семестра и более не появляться, см. раздел А.5.3).

Нормальное изучение курса — поэтапное

Лабораторная работа Лi (приложение Б, регламент — в разделе РЛ) выполняется сразу же после того, как её тема рассказана на лекции. Возникшие вопросы разрешаются на лабораторном занятии, на семинаре или после следующей лекции; после их разрешения следует защита Лi.

Баллы (базовые и бонусные) за лабораторные работы характеризуют *процесс* изучения ОТИК в течение семестра (с 1 по 16 неделю). Баллы за семинары также характеризуют процесс изучения, а не только правильность ответов.

Если оценка $Q(S)$ по ним не соответствует знаниям студента — можно её изменить, оценив *результат* (раздел А.5).

Авральное изучение не делится на этапы

Если студент приступил к изучению курса ОТИК на 13 неделе или позже, процесс изучения невозможно разделить на сколько-нибудь длительные этапы. Сдавать лабораторные работы в такой ситуации **нельзя.**

В этом случае оценивается *результат* изучения по одному из альтернативных вариантов (раздел А.5). Если при подготовке возникли сложности — вопросы также необходимо разрешить на лабораторном занятии, на семинаре или после лекции.

А.2.1. Занятия в течение семестра

Практика показывает, что если с кодами по умолчанию, изучаемыми в курсе ОЭВМ [и Асм], студенты могут более или менее успешно разобраться самостоятельно — то с кодами и методами сжатия ОТИК у тех, кто не ходил ни на лекции, ни на семинары, возникают большие сложности.

Статьи, как правило, иллюстрируют метод на примере *одного* реализующего его алгоритма и одного кода, а в вариантах лабораторных/контрольных работ есть *разные* коды и соответствующие им алгоритмы, реализующие один и тот же метод.

Лекции

Посещаемость поточных лекций не оценивается.

Материалы лекций выложены в `https://gitlab.com/illinc/otik` — этого года — в корне, прошлых лет — в соответствующих поддиректориях.

Семинары

Посещаемость семинара оценивается только если занятие действительно проведено как семинар, а не выродилось в индивидуальную консультацию.

Задачи для семинаров для подготовки к ним вписаны в презентации лекций. Решения с уже прошедших семинаров верстаются и выкладываются только если попросит тот, кто решал, причём *до* стирания с доски (файл `sem44-rs_2022.pdf` — пока единственный такой случай).

Пропущенные даже по уважительной причине семинары «отработать» нельзя; можно придти с другой группой. Вопросы по пропущенным темам задавайте на консультациях, баллы добирайте альтернативными вариантами (раздел А.5).

Лабораторные занятия

На ближайшем занятии после самостоятельной работы над Лi необходимо:
— разрешить с преподавателем все вопросы, которые возникли при подготовке;
— защитить выполненные задания Лi.

Консультации

На 1–16 неделях каждый преподаватель регулярно проводит общие консультации. Расписание и надо ли записываться заранее — уточняйте у преподавателя.

На консультациях можно как задавать любые вопросы по курсу ОТИК, так и сдавать лабораторные работы, но приоритет у обсуждения ВКР/диссертаций выпускных курсов.

А.3. Экзамен/зачёт

Экзамен обязателен для всех студентов. Зачёт у НБ-3* проводится аналогично экзамену у ПИН-3* и тоже обязателен, поэтому не описывается отдельно.

Экзамен/зачёт для каждого студента проводится в одном из двух режимов:

а) блиц-режим (базовые вопросы — необходимо знать первую теорему Шеннона в формулировке для сжатия и характеристики изучаемых кодов, а также уметь делать из этого логические выводы);

б) полный режим (базовые вопросы + вопросы и задачи из билета).

Дополнительные вопросы/задачи могут быть заданы всем и всегда, причём любые — но в полном режиме, как правило, задаются чаще и сами они пространнее.

Блиц-режим а) возможен на досрочном экзамене (при наличии допуска, но независимо от суммы баллов S) или в сессию при $S \geqslant 49$ или с Кр1+Кр2. Полный режим б) возможен только в сессию: либо при $S < 49$, либо по желанию студента.

К экзамену/зачёту **по расписанию допускаются все** (в том числе возможно повышение оценки с нуля до «3 с натяжкой». В этот день разрешаются все сложные вопросы. О досрочном см. раздел А.3.2.

Ответ оценивается не на баллы, а на оценку «неуд» $\leqslant Q_{\text{Итог}} \leqslant$ «отл», баллы добавляются до необходимого значения S (см. раздел А.1.2).

Зачтение статей из Интернета и даже текста лекций ответом на вопрос не считается; оценивается как «неуд». Говорите своими словами. Поясняйте сказанное на примерах.

А.3.1. Билеты и базовые вопросы

Списка базовых вопросов нет и не будет, так как их цель — проверить вашу логику, а не способность к заучиванию. Список билетов выложен в https://gitlab.com/illinc/otik и мало меняется с годами.

А.3.2. Досрочная сдача экзамена/зачёта

Студенты, допущенные к досрочному экзамену/зачёту (таблица А.1), отвечают в блиц-режиме (только на базовые вопросы). Для допуска — необходимо выполнить лабораторные работы **по графику** в указанном объёме.

Также к досрочному экзамену/зачёту допускаются студенты, успешно выполнившие курсовую работу (раздел А.5.3) или индивидуальное задание (А.5.2).

Допуск к досрочному экзамену/зачёту

Таблица А.1

	Недели	Дата	Допуск
Экзамен ПИН-3* (180/5) — ОТИК			
Досрочный-1	15–16	в день последней лекции	Л1–Л5 и зачётка
Досрочный-2	17–18	назначается лектором	Л1–Л4 и зачётка
Плановый	сессия	по расписанию экзаменов	—
Зачёт НБ-31 (108/3) — ОТИК			
Досрочный	15–16	назначается преподавателем	Л1–Л2 и зачётка
Плановый	17–18	назначается преподавателем	—

Для НБ оценка осенью не выставляется в ведомость ОТИК, но записывается и учитывается весной в ведомости ОЭВМ (ОТИК НБ включает часть тем ОЭВМ).

То, что студента устраивает оценка по баллам — не является достаточной причиной получать эту оценку досрочно.

NB: если допуска к досрочному экзамену нет, но почему-то очень хочется (например, в день экзамена нужно быть в каком-то другом месте) — лучше говорите истинную причину: я её по крайней мере обдумаю (но не обязательно дам допуск — принять экзамен досрочно у всех я физически не смогу).

А.4. Долги и пересдачи

Итоговую оценку $Q_{\text{Итог}} =$ «неуд» можно повысить до конца сессии — новая оценка попадает в дополнительную ведомость; иначе образуется **долг.**

Закрыть долг можно **до конца следующего семестра.** Для этого **заранее, до** официальной даты пересдачи, указанной в ОРИОКС, необходимо найти преподавателя, получить и выполнить задание.

Начиная со второго после экзамена семестра (следующая весна и позже) долг не может быть закрыт без письменного разрешения ответственного кафедры.

Для закрытия долга необходимо:

— либо (на «неуд» $\leqslant Q_{\text{Итог}} \leqslant$ «удовл») написать Кр1 и Кр2 (раздел А.5.1);

— либо (на «неуд» $\leqslant Q_{\text{Итог}} \leqslant$ «отл») получить и выполнить индивидуальное задание (раздел А.5.2).

Должник не имеет права сдавать тот же набор лабораторных работ, что и его однокурсники ранее, если только преподаватель не даст ему такое задание.

А.4.1. Долг ОЭВМ [и Асм] с весны

Для закрытия долга ОЭВМ [и Асм] необходимо написать Кр0 (КрЭВМ) и ответить на блиц-вопрос ОЭВМ [и Асм].

А.5. Альтернативы лабораторным работам

По результатам защиты каждого из вариантов выставляется **итоговая оценка за весь курс** «неуд» $\leqslant Q_{\text{Итог}} \leqslant Q_{\max}$ (баллы S подгоняются под оценку $Q_{\text{Итог}}$, раздел А.1.2; все обязательные графы «Лi» маркируются $s = 0{,}01$).

В исключительных случаях, если исполнение и защита безукоризнены, а причина опоздания уважительна — допускается $Q_{\text{Итог}} = Q_{\max} + 1$: «хор» для Кр1 и Кр2 и «хор»/«отл» для индивидуального задания на «удовл»/«хор».

А.5.1. Кр1 (КрХф) и Кр2 (КрRLE/LZ)

$Q_{\max} =$ «удовл» за обе совокупно. По отдельности — оцениваются на баллы соответственно качеству КР и ёмкости графы.

Задания выложены в `https://gitlab.com/illinc/otik`; вариант назначается преподавателем (как из выложенных в `https://gitlab.com/illinc/otik`, так и индивидуальный). Можно писать командой. Ответ без решения = «неуд».

Студентам НБ также необходимо написать две контрольные работы (отличающиеся от Кр1 и Кр2) в соответствии с их семинарами.

Написать Кр1 и Кр2 **на экзамене нельзя** — практика показывает, что это чрезвычайно затягивает экзамен. Задания, аналогичные Кр1 и Кр2, присутствуют

в билетах ОТИК, но выполняться они должны быстро, а защищаться — немедленно после выполнения, с устными пояснениями.

Дистанционные КР для ПИН-41Д, курс ОТИК
— $Q_{\max}$ = «удовл»: необходимо написать Кр0 (ЭВМ) и Кр1 (Хф);
— $Q_{\max}$ = «отл»: все три — Кр0 (ЭВМ), Кр1 (Хф) и Кр2 (RLE/LZ).
Задания совпадают с очными. Обратите внимание: Кр0 (ЭВМ) является частью очного курса ОЭВМ и находится в соответствующем репозитории (см. ссылку):

Кр0 (ЭВМ): https://gitlab.com/illinc/gnu-asm/-/raw/main/kr_0_evm_codes.pdf
Кр1 (Хф): https://gitlab.com/illinc/otik/-/raw/master/kr_1_hf_compress_letters.pdf
Кр2 (RLE/LZ): https://gitlab.com/illinc/otik/-/raw/master/kr_2_lz_compress_chains.pdf

вариант назначается преподавателем по запросу. Запрос — письмо с темой (заголовком, Subject) вида:
— ПИН-41Д Кр0(ЭВМ) <ФИО>
— ПИН-41Д Кр1(Хф) <ФИО>
— ПИН-41Д Кр2(RLE/LZ) <ФИО>
после получения запроса я постараюсь прислать вариант КР вечером (около 24 часов) в день получения заявки; если вам удобнее другой день — укажите это в теле письма. Решение ожидаю увидеть в течение суток после отправки варианта (ответом на то же письмо).

Решение, присланное через двое суток и позже, не засчитывается! Если по какой-либо причине не получается решить вовремя — разберитесь с причиной и позже возьмите другой вариант. Если я не успею прислать вариант в удобный вам день — аналогично: на другой день берите другой вариант.

Дистанционные КР выполняются только индивидуально (исключение — команда, физически находящаяся в одном месте). Так как дистанционные студенты сами выбирают дату написания КР — оценка снижается за опоздание аналогично лабораторным (таблица А.2).

Студенты очных групп, согласно требованиям МИЭТ, должны учиться очно. В виде исключения преподаватель может принять у кого-то КР дистанционно, но не обязан это делать.

А.5.2. Индивидуальные задания

$Q_{\max} \in \{$«удовл», «хор»$\}$, в зависимости от сложности (графа «Индивидуальное задание», если её нет — «Бонус (прочее)»). Выбирается, если студент приступил к занятиям на 12 неделе и позже (когда уже поздно начинать лабораторные или курсовую), но при этом хочет итоговую оценку выше тройки.

Задания на остаток семестра

Объём задания должен соответствовать оставшемуся от семестра времени; на 17 неделе и позже — не более объёма двух лабораторных работ.

Блиц-задания

Сложность блиц-заданий на порядок меньше сложности тех, что даются на остаток семестра, но выполнить их требуется немедленно, не выходя из аудитории.

А.5.3. Курсовая работа (командная или индивидуальная)

Так как мне надоело получать в конце сессии нечто, что не тянет даже на лабораторную работу, но предлагается как «курсовая» — этот путь для ОТИК закрыт. Желающие меня переубедить — подходите лично, с планом конкретной работы.

А.6. Замечания и дополнения

Замечания и дополнения к данному документу можно отправить в письменном виде по адресу https://gitlab.com/illinc/otik/issues.

Принятое замечание/дополнение приносит первому приславшему его студенту от 1 до 8 бонусных баллов.

А.7. Обновление пособия

Между семестрами задания и регламент лабораторных работ качественно обновляются. Выполнять лабораторные работы прошлого года — **нельзя.**

В течение семестра:

— уточняются **формулировки** заданий и регламента, если исходные вызывают недопонимание; суть заданий и регламента — неизменна до конца семестра;
— появляются **новые бонусные** задания;
— дополняется теория (в том числе по вашим замечаниям и дополнениям);

поэтому периодически скачивайте актуальную версию пособия из репозитория https://gitlab.com/illinc/otik. Лабораторные работы в ОРИОКС — неактуальны по определению из-за сложной процедуры обновления. Прямая ссылка на актуальную версию: https://gitlab.com/illinc/otik/-/raw/master/otik-labs.pdf

А.8. ОТИК для ПИН-41Д и дистанционных индивидуальщиков

Дистанционные консультации по ОТИК проводятся по переписке или в видеоконференции, заявки принимаются на электронную почту. Всё, что непонятно в заданиях или теории — **спрашивайте до** написания лабораторных и контрольных работ, а не пытайтесь угадать или, ещё хуже, замаскировать!

Лекций в дистанционном курсе ОТИК не предусмотрено; слайды лекций очного курса — файлы `0ij_тема.pdf` в https://gitlab.com/illinc/otik.

Семинары могут проводиться также в видеоконференции, если поступят заявки на это от не менее чем половины группы. Если заявок будет меньше или не получится согласовать удобное для всех время — семинаров не будет.

Плановые даты защиты лабораторных и написания контрольных работ указаны в ОРИОКС. Если лабораторная или контрольная работа сдаётся с опозданием — максимально возможный балл снижается (таблица А.2).

График снижения баллов за лабораторные и контрольные работы ПИН-41Д

Таблица А.2

Неделя	max Л0$^{\text{НД}}$	max Л1	max Кр0 (КрЭВМ)	max Кр1 (КрХф)	max Кр2 (КрLZ)
1, 2	24				
3, 4	24	24			
5, 6	24	24	24		
7, 8	22	24	24	24	
9, 10	20	22	22	24	24
11, 12	18	20	22	24	24
13, 14	—	18	20	22	24
15, 16	—	—	18	20	22
15, 16 (с нуля) 17, 18, сессия	л/р и КР не принимаются — выполняйте индивидуальное задание				

Вариант лабораторных работ рассчитывается по номеру пропуска (раздел РЛ.2), вариант контрольных — запрашивается у преподавателя (раздел А.5.1). Лабораторные работы в любом случае необходимо защищать (см. разделы РЛ.6 и РЛ.5). Если к решению контрольной работы у преподавателя возникнут вопросы, на них нужно ответить. Оценка выставляется после разрешения всех вопросов.

Приложение Б. Лабораторный практикум по основам теории информации и кодирования

Регламент курса в целом описан в приложении А.

РЛ. Регламент лабораторных работ

Баллы (базовые — график снижения в таблице РЛ.1 — и бонусные) за лабораторные работы Л*i* характеризуют *процесс* изучения ОТИК в течение семестра, поэтому их можно получить, только сдавая Л*i* **по графику:**

— начиная с первого или второго лабораторного занятия (Л1 и Л2 могут выполняться даже до первой лекции, они не требует специфических знаний);

График снижения базовых баллов за лабораторные работы

Таблица РЛ.1

ОТИК ПИН-3*								
Неделя	max Л1	max Л2	max Л3	max Л4 (упрощ.)	max Л5 (упрощ.)	max Л6 (упрощ.)	max 3–4 шт. сразу	max $\geqslant$ 5 шт. сразу
1, 2	7						3	1
3, 4	7	7					3	1
5, 6	6	7	7				3	1
7, 8	5	6	7	10 (5)			3	1
9, 10	4	5	6	10 (5)			3	1
11, 12	—	4	5	9 (4)	10 (5)		3	1
13, 14	—	—	4	8 (3)	10 (5)		1	1
15, 16	—	—	—	7 (2)	9 (4)	10 (5)	1	1
15, 16 (с нуля) 17, 18, сессия	л/р не принимаются — пишите Кр1+Кр2 или выполняйте индивидуальное задание							

ОТИК НБ-3*						
Неделя	max Л0	max Л1	max Л2	max Л3	max 3 шт. сразу	max $\geqslant$ 4 шт. сразу
1, 2, 3, 4	16				3	1
5, 6, 7, 8	16	16			3	1
9, 10, 11, 12	—	16	16		3	1
13, 14, 15, 16	—	—	16	16	1	1
13–16 (с нуля) 17, 18, сессия	л/р не принимаются — пишите Кр0+Кр1 или выполняйте индивидуальное задание					

ОТИК ПИН-41Д — см. таблицу А.2

— по одной работе на каждом занятии, максимум — две на одном занятии;
— с минимальным опозданием.

По результатам защиты лабораторной работы Лi выставляются:

а) на 1–16 неделях по графику (раздел РЛ.5):
   — базовые баллы $0 \leqslant s \leqslant s_{\max}$ ~~в графу «Лi»~~;
   — бонусные баллы $0 \leqslant s^{\text{bonus}} \leqslant s^{\text{bonus}}_{\max}$ ~~в «Бонус (л/р)»~~ (раздел РЛ.4);
   в графу «Лi» выставляется сумма $s + s^{\text{bonus}}$;
б) на 1–16 неделях в составе комплекта из трёх и более работ (раздел РЛ.7):
   — баллы $0 \leqslant s \leqslant 1$ в графе «Лi»;
   — нет бонусных баллов: $s^{\text{bonus}} = 0$;
в) 17–18 недели, сессия — лабораторные работы не принимаются (раздел РЛ.8).

Базовые s и бонусные s^{bonus} баллы выставляются преподавателем:
— не за программу или отчёт, а за **защиту** лабораторной работы;
— на его усмотрение: если описанных явно бонусов/штрафов недостаточно для адекватного оценивания конкретной защиты — вводятся дополнительные;
— сообразно **качеству** защиты и работы: s при $0 \leqslant s \leqslant s_{\max}$ может быть и 1, и 0.

Из баллов Лi, просроченной более чем на две недели без уважительной причины, вычитается величина опоздания (таблица РЛ.1). Просроченная на $\Delta t \geqslant 12$ недель даже по уважительной причине — не может быть сдана вообще.

РЛ.1. Командная работа

Состав команды: либо один студент, либо 2–3 сидящих рядом студента. Задания для $n \in \{1, 2\}$ одинаковы; для $n \geqslant 3$ есть дополнительные (для троек). Независимо от количества участников n, команда выполняет **один вариант** работы (по №, раздел **??**), один отчёт и **одну защиту,** получает **одну оценку.**

Каждый из соавторов должен уметь **объяснить все результаты** лабораторной работы и модифицировать свою часть кода. Если студенты выполняют задания не совместно, а распределяют их между собой, то после того, как решения будут начерно готовы, каждый студент должен:

а) изучить все решения, выполненные другими соавторами;
б) спросить у автора каждого решения всё то, что ему непонятно в его решении — а автор должен объяснить;
в) исправить ошибки в этом решении, если заметит их.

Студента, который не только не пришёл на защиту Лi, но и не помогал её делать, присутствующие соавторы могут временно (на Лi) или постоянно исключить из команды (таблица РЛ.2).

Команды из $n \geqslant 4$ студентов запрещены на ВЦ МИЭТ; в не-ВЦ классах (и задания для $n \geqslant 4$ сверх заданий для троек) — на усмотрение преподавателя.

Командная работа и сложные ситуации

Таблица РЛ.2

	Нормально	**Недопустимо**
Изменение состава команды (постоянное)	Один-два раза (не сработались) — баллы за сданные Лi сохраняются. № команд д. б. уникальным — утвердите № у преподавателя, прежде чем делать новый вариант	Постоянно делиться/объединяться синхронно с наличием/отсутствием заданий для троек — штраф ко всем Лi по -2 балла к каждой
Защита Лi в неполном составе (подготовка Лi — всегда в полном)	Отсутствовать на защите некоторых Лi — защищаться отдельно не нужно. Тройка выполняет задания для троек, даже если на защите один студент	Отсутствовать систематически (и не сообщать о сложностях): у пропавшего на полсеместра без предупреждения — аннулируются баллы за все сданные без него Лi
Исключение из команды на одну Лi (временное)	Делать и защищать с командой Л$(i+1)$ и следующие. На тему Лi — получить и выполнить доп. задание или смириться с $s(\text{Л}i) = 0$	Защищать ту же самую Лi, которую уже защитила команда — в лучшем случае будет $s(\text{Л}i) = 1$
Постоянное исключение из команды — изменение состава; исключённый меняет №		
Разные мнения на защите Лi	Придти к согласию, затем отвечать. Не удалось договориться — скажите об этом и изложите все мнения последовательно	Излагать все мнения одновременно — ни один ответ не засчитывается; озлобленность преподавателя растёт
Разные мнения при подготовке Лi	Спросить у преподавателя и только потом защищаться	Всё остальное — штрафуется либо как недоделка, либо как жульничество

РЛ.2. Выбор варианта и номера команды

Вариант в каждом задании рассчитывается по номеру команды (№) — один для всех участников команды.

Распределение № обычно происходит на первом лабораторном занятии. Группа может поделиться на команды и распределить уникальные (в пределах группы) номера заранее — тогда преподавателю лабораторных работ на первом занятии необходимо предоставить список команд, отсортированный по возрастанию №, а внутри каждой команды — по алфавиту.

Отдельно взятая команда не может сама выбрать себе №, так как это не гарантирует уникальность — необходимо утверждение у преподавателя (таблица РЛ.2).

Если студент один в команде — он может использовать №, равный номеру пропуска. За самовольное присвоение №1 без согласования с группой и преподавателем — штраф -2 балла к первой по графику лабораторной работе.

РЛ.3. Требования к выполнению лабораторных работ

Лабораторные работы могут быть подготовлены дома и защищаться на ВЦ МИЭТ. Перед защитой на лабораторном занятии необходимо разрешить все вопросы, возникшие в процессе подготовки.

Средства разработки

Любые, позволяющие работать с «сырыми» бинарными данными, в том числе заданной разрядности. Если вы записываете в файл два байта — шестнадцатеричный просмотровщик/редактор должен показать, что там именно два байта, причём именно те, что вы записывали.

Если в желаемом языке есть нужные средства, но вы ими не владеете — либо изучайте их сейчас, либо пишите на C/C++, коллекция компиляторов GCC.

Отчёт

Дистанционные группы оформляют отчёт в любом случае. Для очных групп:
— если Лi сделана на занятии — защищайте её сразу, **не оформляя отчёта;**
— если Лi готовится дома заранее — параллельно оформляйте отчёт.
Формат — OpenDocument или PDF. Заголовок отчёта должен включать имя группы и ФИО авторов, тему работы, а также — для каждого задания:
— номер и текст задания;
— номер и текст варианта (если есть);
— **платформа,** на которой делается задание, в частности — ОС и разрядность;
— ключевые фрагменты программного кода;
— ключевые пояснения, которые вы боитесь забыть;
— ссылки на использованные справочные материалы и пояснения, что и для чего вы там искали, где применили найденное;
— результат выполнения задания: результаты измерений с комментариями.
Вместе с отчётом должен предоставляться полный текст программ, готовый к сборке и запуску, поэтому копировать их ещё и в отчёт не нужно.

РЛ.4. Необязательные (бонусные) задания (1–16 недели)

Задания, отмеченные как **«Бонус»**, *необязательны.* На 1–16 неделях за их выполнение начисляются дополнительные баллы $0 \leqslant s^{\text{bonus}} \leqslant s^{\text{bonus}}_{\max}$ (максимальное количество $s^{\text{bonus}}_{\max}$ указано в тексте задания и не уменьшается со временем: s^{bonus} зависит только от качества исполнения и защиты, но не от времени сдачи).

Баллы s^{bonus} добавляются к графе ~~«Бонус (л/р)»~~ «Лi». Бонусные задания могут быть сданы либо одновременно с соответствующей лабораторной работой, либо

(при условии, что это не помешает преподавателю принимать у других студентов обязательные задания) после неё.

При защите нескольких лабораторных работ сразу (раздел РЛ.7) бонусные баллы не начисляются ($s^{\text{bonus}} = 0$), бонусные задания формируют базовые баллы s наравне с обязательными.

РЛ.5. Защита лабораторной работы (1–16 недели)

До начала защиты лабораторной работы Лi команда должна задать преподавателю подготовленные во время выполнения Лi вопросы.

Если у вас вопрос — говорите «у меня вопрос», если планируете защищаться — «хочу сдать»/«хочу защитить работу». Запрос «посмотрите лабораторную работу» будет по возможности игнорироваться, так как его цель не ясна.

Дополнительные вопросы на защите задаются всем. **Отчёта для оценивания недостаточно.** По итогам защиты ставятся баллы $s + s^{\text{bonus}}$.

Защищать не по порядку — можно, $\max(s_{\max})$ **корректируются**

Если какую-то Лi не удалось выполнить даже частично:
— сформулируйте вопросы к преподавателю о том, что вызвало проблемы в Лi;
— если Л$(i+1)$ сделать получается — делайте и защищайте;
— на том же занятии задавайте вопросы про Лi;
$\max(s_{\max})$ считается по фактическому порядку защит, а не по номерам $i/(i+1)$.

Не выполненная, но проработанная Лi + дополнительное задание = «удовл»

Если ни одно задание лабораторной работы Лi не выполнено, но:
— по всем обязательным заданиям Лi подготовлены вопросы к преподавателю;
— сохранены все неудачные попытки выполнения заданий с комментариями, чем именно они плохи (не собирается — с какими сообщениями компилятора? Не приводит к нужному результату — а к какому приводит?);
— после разрешения вопросов команда выполнила по указанию преподавателя либо задание Лi, либо дополнительное задание, не покидая аудиторию;
то Лi засчитывается на $1 \leqslant s \leqslant \frac{1}{2}\max(s_{\max})$, без бонусных баллов ($s^{\text{bonus}} = 0$).

Чего делать не стоит («неуд», то есть 0 баллов)

Однозначно на «неуд» (на $s = s^{\text{bonus}} = 0$ баллов) оценивается работа Лi, если вопросов перед защитой не задавалось, зато на самой защите:
— звучат ответы «делал давно, не помню», «взял в Интернете, не помню где»;
— есть хотя бы одна программа (в том числе отлично выполненная), в которой студенты ничего не могут объяснить и/или изменить;
— в отчёте есть хотя бы одно место, которое студенты не могут разъяснить и пересказать своими словами;

— хотя бы в одном задании Лi (основном или бонусном) есть признаки плагиата (несоответствие варианту, несоответствие программы заявленным ОС и разрядности, использование заданий предыдущего года и ранее, и т. п.);

как только хоть что-то из этого проявилось — защита завершается. Преподаватель может как дать команде дополнительное задание на $1 \leqslant s \leqslant 2$ (но $s^{\text{bonus}} = 0$), так и сразу отправить на пересдачу (на следующем занятии, тому же преподавателю).

РЛ.6. Дистанционная защита для дистанционных групп

Дистанционные группы защищают лабораторные работы, используя:

— синхронную связь с демонстрацией экрана (телемост=видеоконференция) — регламент и требования к отчёту полностью аналогичны очной защите;
— или асинхронную (почта, домашние задания ОРИОКС) — отчёт необходим, но оценивается защита; **«прислать лабы на почту»** без защиты **= «неуд».**

«На одном занятии» дистанционным группам читать как «за одну неделю из 1–16», «на следующем занятии» — как «через две недели».

Студенты очных групп, согласно требованиям МИЭТ, должны учиться очно. В виде исключения преподаватель лабораторных работ может принять у кого-то защиту дистанционно, но не обязан это делать.

РЛ.7. Защита двух, трёх или более лабораторных работ (1–16 недели)

Две лабораторные работы на одном занятии могут быть оценены раздельно (как описано в разделе РЛ.5); если есть очередь на защиту — после защиты первой команда отправляется в конец очереди.

Комплект (три и более работы сразу) — оценивается выборочно, без бонусов

Если команда приносит на занятие три и более лабораторных работы сразу — защита происходит выборочно: преподаватель проверяет несколько произвольно выбранных (из всех выполненных, в том числе бонусных) заданий разных работ, задаёт вопросы и даёт задания по произвольным темам.

В каждую графу «Лi» выставляются одинаковые баллы:

$$\begin{cases} 0 \leqslant s \leqslant 3, & \text{если в комплекте 3–4 работы (до 12 недели),} \\ 0 \leqslant s \leqslant 1, & \text{если работ} \geqslant 5 \text{ или неделя} \geqslant 13. \end{cases} \tag{Л.1}$$

Бонусные баллы, которые рассчитываются для одной лабораторной работы, при защите комплекта не рассчитываются и не добавляются: $s^{\text{bonus}} = 0$.

Третья лабораторная работа без предупреждения — ждёт две недели

Если команда уже получила баллы за две работы (или за комплект из 3–4) на занятии — защита следующей не ранее следующего занятия или через две недели; регламент — по фактической дате защиты, а не по первой заявке.

РЛ.8. Лабораторные работы на 17–18 неделях и в сессию

Раздельная защита лабораторных работ (с вычислением баллов, бонусов и штрафов к каждой работе отдельно) на 17–18 неделях и в сессию **невозможна.**

Если, вопреки требованиям регламента, команда принесёт на 17 неделе и позже любое количество лабораторных работ (от одной и более), преподаватель может:
— либо оценить их как один комплект: выборочно, на $0 \leqslant s \leqslant 1$ и $s^{\text{bonus}} = 0$;
— либо сразу послать такую команду писать Кр1 и Кр2.

Исключение — если команда сдавала лабораторные работы весь семестр, начиная с первого занятия, по графику и досдаёт тому же преподавателю *одну* последнюю работу — она оценивается как сделанная на 16 неделе.

Лабораторная работа $0^{НД}$

Представление данных в ЭВМ

Засчитывается только осенью 2025 г.; актуальная версия в https://gitlab.com/illinc/otik

Только для НБ-3* и ПИН-*Д! Изучавшие ОЭВМ и Асм — начинают ОТИК с Л1.

Максимальная оценка ПИН — 0 баллов, НБ — 16 баллов, ПИН-*Д — 24.
Штраф за одно задание вообще без ответов и чисел НБ -4 балла, ПИН-*Д -5.

Обратите внимание, что цель Л$0^{НД}$ — **изучить представление данных в ЭВМ,** а не «написать много кода»! Если студент не ответил на **вопросы,** приведённые в задании — Л$0^{НД}$ не может быть засчитана.
Наоборот, если студент может ответить хотя бы на часть вопросов каждого задания и привести примеры, то Л$0^{НД}$ засчитывается независимо от того, какими инструментами проводилось исследование.
В любом случае **инструменты** должны быть описаны в отчёте, использованные **ссылки** на документацию также помещены в отчёт (для книг указываются не только выходные данные, но и номер страницы; для интернет-источников без деления на страницы — не только URL, но и раздел, и цитата для поиска).

Л$0^{НД}$.1. Рекомендации и требования к работе

1. Все задания сформулированы для C/C++, как для ЯВУ ОТИК по умолчанию — но выполняться могут на любом ЯВУ. Если для работы с файлами ОТИК планируется использовать другой ЯВУ — используйте именно его.
2. В Л$0^{НД}$.№1 *обязательно* для оценки диапазона N-битного значения использовать средства и/или документацию выбранного ЯВУ.
   Штраф -2 балла за каждую лабораторную работу, где чтение/запись в файл выполняется средствами не того ЯВУ, который исследован в Л$0^{НД}$.№1.
3. В Л$0^{НД}$.№2–Л$0^{НД}$.№5 *рекомендуется* использовать выбранный ЯВУ. Но если в нём нет средств для исследования памяти — допускается C/C++.
4. Во всех заданиях Л$0^{НД}$, если они реализуются на C/C++, рекомендуется вывод с помощью функции стандартной библиотеки C $printf()$, так как задание формата вывода для $printf()$:
   а) компактно;
   б) влияет только на одно значение;
   в) полностью определяется программистом.

   Не рекомендуется вывод в потоки (доступен только в C++), так как:
   а) вывод в заданном формате (а не по умолчанию) объёмен и запутан;
   б) большинство манипуляторов не прекращают своё действие до отмены, а в комбинации с другими дают неочевидные эффекты;

в) придётся выполнять лишние преобразования, так как по умолчанию в поток и $char$ / $unsigned\ char$, и $int8\_t$ / $uint8\_t$, и указатели на них выводятся как символы/строки.

Тем не менее, вывод в потоки не штрафуется. Часть заданий различается для $printf()$ и для потоков из-за принципиально разного подхода к выводу.

Л0^{НД}.2. Задание на лабораторную работу

Задание Л0^{НД}.№1. При помощи оператора `sizeof` выясните, сколько байтов занимают переменные типов: `char`, `unsigned char`, `short`, `unsigned short`, `int`, `unsigned`, `long long`, `unsigned long long`, `float`, `double`.

Для каждого типа рассчитайте разрядность в битах, учитывая, что для архитектуры x86/amd64 байт = октет = 8 бит.

Напечатайте, используя <climits> (limits.h) C или <limits> C++, минимальные и максимальные значения исследуемых знаковых и беззнаковых целых типов.

Как сказано выше, если в дальнейшем планируется использовать отличный от C/C++ ЯВУ — то измерения должны выполняться в этом ЯВУ, а не в C/C++. Из стандартных типов используемого ЯВУ выберите аналоги перечисленных; опишите соответствие, размер, минимальные и максимальные значения.

1. Сколько различных значений может принимать переменная беззнакового N-битного типа? Знакового N-битного?
   Связано ли это как-то со значением N и как именно?
2. Каждое ли целое число $x \in [0, 2^N)$ имеет своё N-битное представление? Всякая ли последовательность ξ из N битов может быть рассмотрена как целое $x \in [0, 2^N)$? Взаимно однозначно ли это соответствие?
3. Каждое ли целое число $x \in [-2^{N-1}, 2^{N-1})$ имеет своё N-битное представление? Всякая ли последовательность ξ из N битов может быть рассмотрена как целое $x \in [-2^{N-1}, 2^{N-1})$? Взаимно однозначно ли это соответствие?

Напечатайте минимальные и максимальные значения $\min$ и $\max$ исследуемых типов с плавающей запятой.

1. Каждое ли вещественное число $x \in [\min, \max]$ имеет своё представление с плавающей запятой стандарта IEEE 754 ($float/double$ ЭВМ)?
2. Взаимно однозначно ли соответствие вещественных чисел $x \in [\min, \max]$ и их представлений с плавающей запятой стандарта IEEE 754?
3. Всякая ли последовательность ξ из N битов $\Big(N \in \{32, 64\}\Big)$ может быть рассмотрена как N-битное *значение с плавающей запятой*?
   Всегда ли это значение — *число?*
4. Каких чисел больше: 32-битных целых ($int/unsigned$) или 32-битных с плавающей запятой ($float$)? 64-битных целых ($long\ long/unsigned\ long\ long$) или 64-битных с плавающей запятой ($double$)?

Значения $\pm\infty$ числами не являются, нечисла nan — тем более.

Как в выбранном ЯВУ записать **в файл:**

а) целое число заданной разрядности?

б) число с плавающей запятой заданной разрядности?

Задание Л0$^{\text{НД}}$.№2. Исследуйте внутреннее представление 16-битных чисел. Для этого на C/C++ разработайте $\mathrm{void}\ print16(\mathrm{void}\ *p)$ (одну: С или П).

С-Л0$^{\text{НД}}$.№2 для вывода при помощи стандартной библиотеки C (рекомендуется): функция $print16()$ интерпретирует p как адрес 16-битного целого числа x типа $short/unsigned\ short$ и печатает x при помощи $printf()$:

С-а) в шестнадцатеричном представлении;

С-б) в двоичном представлении: если не поддерживается формат b — напечатать биты `(x & (1 << i)) != 0`, от старшего $i = 15$ и до $i = 0$;

С-в) в десятичном беззнаковом представлении;

С-г) в десятичном знаковом представлении.

Необходимый вид вывода $print16()$ для значений `13` и `0x8000` по адресу p:

```
000D 0000000000001101     13    +13
8000 1000000000000000 32768 -32768
```

Для $print16()$, а также последующих $print32()$, $print64()$, и любых x и y младшая цифра любого представления y всегда должна печататься под младшей цифрой соответствующего представления x.

П-Л0$^{\text{НД}}$.№2 для вывода в потоки (крайне не рекомендуется, но и не штрафуется). Так как вывод в поток различает $short$ и $unsigned\ short$, функция $print16()$ интерпретирует адрес p дважды:

— как адрес 16-битного беззнакового целого ux типа $unsigned\ short$;

— и как адрес 16-битного знакового целого sx типа $short$;

и печатает оба ux и sx во всех доступных представлениях:

П-а) ux в шестнадцатеричном представлении;

П-б) ux в двоичном представлении (шаблон `std::bitset<N>`);

П-в) ux в десятичном представлении;

П-г) sx в шестнадцатеричном представлении;

П-д) sx в двоичном представлении (шаблон `std::bitset<N>`);

П-е) sx в десятичном представлении.

Не забывайте, что манипуляторы dec/hex действуют на все числа до отмены, а $setw(\mathrm{int}\ w)$ — только на одно следующее.

Убедитесь в процессе исследования, что (П-б) и (П-д) — одно и то же двоичное представление; (П-а) и (П-г) — одно и то же шестнадцатеричное представление. Если у вас (П-б)≠(П-д) или (П-а)≠(П-г) — ищите ошибку.

Сократите вывод П-Л0$^{\text{НД}}$.№2 до вида, аналогичного С-Л0$^{\text{НД}}$.№2.

Исследуйте при помощи $print16()$ 16-битные целочисленные переменные типа $short/unsigned\ short$, принимающие значения:

— минимальное и максимальное целое беззнаковое 16-битные значения;
— минимальное и максимальное целое знаковое 16-битные значения;
— целочисленные x, y, a, b, соответствующие варианту (таблица Л0^{НД}.1).

Варианты значений

Таблица Л0^{НД}.1

$(№ - 1)\%2 + 1$	**Вариант**
1	$x = 9, y = -9, a = 1, b = 2, c = 12345678, d = 123456789$
2	$x = 5, y = -5, a = 1, b = 2, c = 12345689, d = 123456891$

Как представляются в памяти ЭВМ знаковые целые значения? Как различаются целочисленные x и $y = -x$?

Как связаны двоичное представление (С-б) и шестнадцатеричное (С-а)? Как по шестнадцатеричному (С-а) записать двоичное для любого выбранного числа?

Задание Л0^{НД}.№3. Исследуйте внутреннее представление 32-битных чисел — целых $int/unsigned$ и с плавающей запятой $float$. Для этого на C/C++ разработайте void $print32$(void $*p$) (также одну— С или П — соответственно Л0^{НД}.№2).

С-Л0^{НД}.№3 для стандартной библиотеки С: $print32()$ интерпретирует p дважды:

— как адрес 32-битного целого числа x типа $int/unsigned$;
— как адрес 32-битного числа с плавающей запятой fx типа $float$;

и печатает x в представлениях (С-а)–(С-г), а fx:

С-д) в шестнадцатеричном экспоненциальном представлении;
С-е) в десятичном экспоненциальном представлении;
С-ж) в представлении с десятичной запятой.

Необходимый вид вывода $print32()$ для значений `13` и `0x80000000` по адресу p:

```
0000000D 00000000000000000000000000001101          13          +13
    +0X1.A0P-146    +1.82e-44         +0.00
80000000 10000000000000000000000000000000 2147483648 -2147483648
      -0X0.00P+0    -0.00e+00         -0.00
```

Если ширина терминала позволяет вывести (С-а)–(С-ж) в одну строку — это необходимо сделать; иначе вторая строка должна иметь отступ.

П-Л0^{НД}.№3 для вывода в потоки: $print32()$ интерпретирует адрес p трижды:

— как адрес 32-битного беззнакового целого ux типа $unsigned$;
— как адрес 32-битного знакового целого sx типа int;

— как адрес 32-битного числа с плавающей запятой fx типа $float$;
и печатает ux и sx в представлениях, аналогичных (С-а)–(С-г) (дублирующиеся — в одном экземпляре), а fx — в представлениях, аналогичных (С-д)–(С-ж). Обратите внимание, что манипулятор $setprecision(\text{int } n)$ действует до отмены; $fixed/scientific/hexfloat$ — также до отмены, а также проявляют себя по-разному в зависимости от dec/hex.

Исследуйте при помощи $print32()$ 32-битные переменные — целочисленные типа $int/unsigned$ и с плавающей запятой типа $float$:
— минимальное и максимальное целое беззнаковое 32-битные значения;
— минимальное и максимальное целое знаковое 32-битные значения;
— целочисленные x, y, a, b, c, d, соответствующие варианту (таблица Л0$^{\text{НД}}$.1);
— $float$-значения x, y, a, b, c, d, соответствующие варианту (таблица Л0$^{\text{НД}}$.1);

Чем различается одно и то же значение (x, y, a, b), записанное в переменную типа $short/unsigned\ short$ и $int/unsigned$? Как различаются граничные (минимальные и максимальные) значения?

Похожа ли структура $int/unsigned$ и $float$? Как представляются значения с плавающей запятой? Как различаются x и $y = -x$ с плавающей запятой?

Получилось ли точно представить d как $float$? Почему? Существует ли такое $float$-значение z, что $d - z = 1$?

Есть ли среди исследованных переменных $float$-числа: нули -0 и $+0$, минимальное денормализованное положительное? Нечисла? Как они записываются?

Как записываются $float$-значения:
— минимальное нормализованное положительное число;
— предыдущее перед $+2$ (максимальное $float$-число x такое, что $x < +2$);
— максимальное нормализованное положительное число;
— $+\infty$?
Запишите предполагаемое битовое представление на бумаге, представьте его шестнадцатерично и инициализируйте им в программе переменную $int/unsigned$. Проверьте $print32()$, что это действительно нужное $float$-значение.

В каком интервале $float$-чисел больше: $[+0, +2)$ или $[+2, +\infty)$?

Задание Л0$^{\text{НД}}$.№4. Аналогично Л0$^{\text{НД}}$.№3, разработайте void $print64(\text{void } *p)$, аналогичную $print32()$ для размера 64 бита.

Необходимый вид вывода $print64()$ для значений 13 и 0x8000000000000000:

```
000000000000000D 0000000000000000000000000000000000000000000000000000000000001101
                                           13                  +13
          +0X0.00P-1022           +6.42e-323                +0.00
8000000000000000 1000000000000000000000000000000000000000000000000000000000000000
                          9223372036854775808 -9223372036854775808
             -0X0.00P+0            -0.00e+00                 -0.00
```

Если ширина терминала позволяет вывести данные в одну или хотя бы в две строки — это необходимо сделать; иначе вторая и третья должна иметь отступ.

Если двоичное представление (С-б) выводится при помощи битовых операций и оно явно некорректно, а в $print32()$ и $print16()$ всё было хорошо — вспомните, что литерал `1` имеет тип int. Единица-литерал типа $long\ long$ — `1ll` или `1LL`.

Исследуйте $print64()$ переменные $long\ long/unsigned\ long\ long$ и $double$:
— целочисленные x, a, соответствующие варианту (таблица Л0[НД].1);
— $double$-значения x, a, соответствующие варианту (таблица Л0[НД].1).
Чем различается одно и то же значение (x или a), записанное в $float$ и $double$? В каком интервале $double$-чисел больше: $[+0,+2)$ или $[+2,+\infty)$?

Задание Л0[НД].№5. Разработайте функцию void $printDump($void $*p$, size_t $N)$, которая преобразует нетипизированный указатель p в указатель $p1$ на байт ($char/unsigned\ char$) и печатает шестнадцатеричные значения N байтов, начиная с этого адреса: $*p1, *(p1+1), ... *\Big(p1+(N-1)\Big)$ — шестнадцатеричный дамп памяти. Каждый байт должен выводиться в виде двух шестнадцатеричных цифр, байты разделяются пробелом (спецификатор «$\%02hhX$ »).

Исследуйте при помощи $printDump()$, как хранятся в памяти компьютера:
а) целое число x (типа int; таблица Л0[НД].1); по результату исследования определите порядок следования байтов в словах для вашего процессора:
   — прямой (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему);
   — обратный (младший байт по старшему адресу, порядок Motorola, Big-Endian, от старшего к младшему);
б) число с плавающей запятой x (типа $double$; таблица Л0[НД].1).
в) строки `"abc012"` и `"абв012"` (массив из $char$; при выборе N учитывайте всю длину строки, а не только видимые буквы);
г) «широкие» строки `L"abc012"` и `L"абв012"` (массив из $wchar\_t$; при выборе N учитывайте всю длину строки).

На MS Windows возможна (если файл исходного кода сохранён в однобайтовой кодировке `windows-1251`) ситуация, когда литерал `L"абв012"` не воспринимается компилятором как корректная широкая строка. В этом случае в отчёте не будет её дампа, но должно быть сообщение компилятора об ошибке.

Каков порядок байтов в вашей ЭВМ? Как хранятся символы и строки в памяти ЭВМ? Чем широкие символы и строки отличаются от узких?

Л0[НД].3. Дополнительные бонусные и штрафные баллы (НБ-*/ПИН-*Д)

$-4/-5$ баллов за каждое задание, где вообще нет ответов на вопросы.

$-3/-4$ балла за каждое задание, где приведены ответы на часть вопросов, но нет иллюстрирующих их чисел (размеров/диапазонов/значений/дампов), то есть вывода разработанных программ или фрагментов документации.

Как для НБ-*, так и для ПИН-*Д: от -2 до -1 балла за каждое задание, где приведены одновременно:

— ответы менее, чем на $\frac{2}{3}$ общего количества вопросов в задании:
  а) -2 балла — менее, чем на треть вопросов;
  б) -1 балл — от $\frac{1}{3}$ до $\frac{2}{3}$ вопросов;
— числа, соответствие заданию (рассчитанные программно, вручную или взятые из документации).

Лабораторная работа 1

Просмотр и редактирование файлов в шестнадцатеричном представлении. Работа с файлами в C/C++

Засчитывается только осенью 2025 г.; актуальная версия в https://gitlab.com/illinc/otik

Максимальная оценка ПИН — 7 баллов.
Штраф за одно пропущенное обязательное задание -2 балла.

Л1.1. Задание на лабораторную работу

Задание Л1.№1. С помощью hexdump/xxd или шестнадцатеричного просмотровщика/редактора исследуйте файлы различных форматов (некоторые файлы представлены в папке «labs-files/Файлы в разных форматах»). Выделите сигнатуры или иные признаки формата там, где это возможно.

Описания части форматов находятся в папке «labs-files/Описание некоторых форматов», для прочих можно найти в Сети.

Задание Л1.№2. Определите тип файла, соответствующего номеру варианта ($(№ - 1)\%10$), из папки «labs-files/Варианты 1 — *». Откройте его корректным приложением. Поместите в отчёт тип и описание содержимого файла, а также признаки формата, по которым удалось определить тип.

Задание Л1.№3. В соответствии с номером варианта отредактируйте (таблица Л1.1) изображение colorchess16x16x2.bmp, используя шестнадцатеричный редактор. Откройте изменённый файл и убедитесь, что изменения корректны.

Файл colorchess16x16x2.bmp представляет собой изображение 16×16 пикселей, сиренево-болотное (две сиреневые и две болотные клетки по 8×8 пикселей), глубина цвета — 1 бит на пиксель. Убедитесь, что просмотровщик отображает его как цветное (некоторые игнорируют палитру файлов с глубиной 1 бит на пиксель).

Варианты действий для редактирования

Таблица Л1.1

$(№ - 1)\%3 + 1$	**Вариант**
1	Поставить зелёную точку в правом нижнем углу изображения
2	Поставить сиреневую точку в правом верхнем углу изображения
3	Поставить зелёную точку в левом верхнем углу изображения

Задание Л1.№4. С помощью hexdump/xxd или шестнадцатеричного просмотровщика/редактора исследуйте файлы формата «простой текст» (plain text),

представленные в различных кодировках (папка `labs-files/Файлы в формате простого текста - кодировки разные`, раздел Л1.2).

Есть ли у простого текста заголовок?

Сравните один и тот же текст, представленный в различных кодировках: размер и шестнадцатеричное представление.

Сравните шестнадцатеричное представление с тем, как текстовый редактор читает этот текст в кодировке платформы (обычно UTF-8). Учтите, что малофункциональные редакторы, такие как Блокнот MS Windows, поддерживают только одну-две кодировки, и «кракозябры» для остальных — нормальное явление.

Обратите внимание на файлы `Алфавит - *`, включающие 192 символа национальных кодовых таблиц русского языка / кодовой таблицы Unicode, в том числе:

— 189 печатных (пробелы, цифры, латинские и русские буквы, символ @);

— 3 управляющих (переводы строки LF в стиле UNIX).

Одинаково ли количество байтов для представления одного символа кодовой таблицы (печатного или управляющего, как LF) в различных кодировках?

Одинаково ли шестнадцатеричное представление пробела в различных кодировках? Цифр? Латинских букв? Русских букв?

Задание Л1.№5. Для ПМ-3* и ПИН-*Д — бонус +1 или +10 баллов; для очных групп ПИН-3* 0 баллов: включено в Л2, здесь не засчитывается. Для файла $(№ - 1)\%9$ из папки `labs-files/Варианты 2 - определение кодировки простого текста` (далее — файл W):

— определите, является ли W простым текстом на русском языке в одной из стандартных кодировок (один из вариантов представляет собой нерусскоязычный текст);

— если да — определите кодировку и декодируйте в UTF-8.

Если для определения кодировки используется существующая программа определения кодировок, задание засчитывается на $+1$ балл. Для получения $+10$ баллов необходимо провести частотный анализ самостоятельно.

Для проведения частотного анализа выполните следующие шаги.

1. Разработайте программу для определения частот октетов (байтов x86) в заданном файле (это может быть как скрипт-однострочник, использующий стандартные утилиты GNU/Linux, так и проект на любом языке программирования в любой среде).
   Хотя в этом задании далее анализироваться будут файлы в формате простого текста — программа, анализирующая распределение октетов, *должна корректно обрабатывать любые файлы.*

2. Рассчитайте частоты появления октетов в файлах, являющихся осмысленным русскоязычным текстом достаточного объёма в различных кодировках (`labs-files/Файлы в формате простого текста - кодировки разные`).

Определите:
- — четыре наиболее частых октета среди всех используемых;
- — четыре наиболее частых октета, не являющихся кодами печатных символов ASCII; для однобайтовых кодировок сопоставьте соотношение их частот с частотами символов русского языка.

Обратите внимание на распределение октетов многобайтовых кодировок Unicode (UTF-8, UTF-16, UTF-32).

3. Рассчитайте частоты появления октетов в файле W. Определите, аналогично п. 2, четыре наиболее частых октета среди всех и четыре — среди не являющихся кодами печатных символов ASCII.
   Сопоставьте их с результатами п. 2 и с частотами символов русского языка. Определите наиболее вероятную кодировку или нерусскоязычность текста.
4. Если по результатам п. 3 файл W является русскоязычным текстом в кодировке X — декодируйте W из X в UTF-8 любой утилитой перекодировки. Проверьте корректность результата.

Л1.2. Кодировки и кодовые таблицы русского языка

Кодировки и кодовые таблицы

Строго говоря, необходимо различать понятия:
- — *кодовая таблица* или таблица кодов — соответствие символов кодам в каком-то диапазоне;
- — *кодировка* — представление кода символа в памяти или на диске.

Но обычно их смешивают и называют для краткости «кодировкой» совокупность кодовой таблицы и собственно кодировки.

Это в большинстве случаев не приводит к недопониманию, так как:
- — кодировкам UTF-8, UTF-16, UTF-32 всегда соответствует одна и та же универсальная кодовая таблица Unicode, содержащая *все* национальные алфавиты;
- — национальным кодовым таблицам (KOI8-R, IBM CP866, Windows-1251 и т. п.) всегда соответствует одна и та же кодировка: код записывается октетом «как есть».

Таким образом, *однобайтовыми кодировками* на практике называются *кодовые таблицы,* сопоставляющие символы кодам в диапазоне 0–255 (0x00–0xFF); коды символов таких таблиц всегда записывается одним октетом (байтом x86). В настоящее время все такие кодовые таблицы сопоставляют кодам 0–127 те же символы, что и кодовая таблица ASCII (являются расширениями кодовой таблицы ASCII). Коды в диапазоне 128–255 описывают национальные кодовые страницы.

Кодовая таблица ASCII сопоставляет символы кодам в диапазоне 0–127 (0x00–0x7F) (см. приложение В). При этом понятие «однобайтовая кодировка ASCII» *не определено,* и в зависимости от контекста может описывать как исторические способы записи семибитных ASCII-кодов восемью битами (старший бит

мог использоваться для контроля чётности, дублировать один из семи младших или всегда быть нулевым), так и Latin-1 (ISO 8859-1), и, чаще, ту однобайтовую кодировку, которая используется на компьютере говорящего.

Кодовая таблица Unicode сопоставляет символы кодам `0`–`0x10 FFFF` (кодам 0–127 соответствуют символы ASCII) то есть не может быть представлена однобайтовой кодировкой. Не всем Unicode-кодам соответствуют символы; так, коды `0xD800`–`0xDFFF` зарезервированы для представления суррогатных пар в UTF-16.

Для кодирования символов Unicode используются три основные кодировки.

1. UTF-8, кодировка переменной длины (изначально от 1 до 6 октетов, позже ограничили до 4) для узких строк:
   — `0xxx xxxx` — ASCII-символы (коды от 0 до $127 =$ `0x7F`) представляются одним байтом, равным ASCII-коду (старший бит — 0).

   Прочие, включая 128–255, представляются не менее чем двумя байтами. Старшие биты первого из k байтов содержат k единиц, затем следует разделитель 0, затем — старшие биты Unicode-кода. Старшие биты последующих байтов — 10, так что представление символа в UTF-8 не равно его коду Unicode:
   — `110x xxxx 10xx xxxx` — символы с Unicode-кодами от $128 =$ `0x80` до `0x7FF`, в том числе греческие, русские и арабские буквы, представляются двумя байтами;
   — `1110 xxxx 10xx xxxx 10xx xxxx` — далее символы до `0xFFFF`, в том числе основные китайские и японские иероглифы — тремя;
   — `1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx` — прочие символы (с запасом — до `0x1F FFFF`) могут быть представлены четырьмя байтами.

   UTF-8 — наиболее распространённая в настоящее время кодировка Unicode.
2. UTF-16, кодировка переменной длины (от 1 до 2 элементов) для широких строк из двухоктетных (16-битных) элементов:
   — символы с Unicode-кодами `0x0000`–`0xFFFF` записываются одним 16-битным элементом «как есть»; символов с кодами `0xD800`–`0xDFFF` не существует;
   — символы `0x1 0000`–`0x10 FFFF` — двумя элементами: первый лежит в диапазоне `0xD800`–`0xDBFF`, второй — `0xDC00`–`0xDFFF` (суррогатной парой);

   UTF-16 — старейшая кодировка Unicode (первые версии Unicode умещались в один 16-битный элемент). UTF-16, в отличие от UTF-8 и UTF-32, не позволяет записать коды свыше `0x10 FFFF`.
3. UTF-32, кодировка постоянной длины (один 32-битный элемент с запасом вмещает все Unicode-коды) для широких строк из четырёхоктетных (32-битных) элементов.

Кодировки UTF-16 и UTF-32 имеют варианты, соответствующие разному порядку байтов в двух- или четырёхоктетном элементе (LE/BE); по умолчанию подразумевается порядок байтов платформы (LE для x86).

В MS Windows «Unicode» обозначает устаревшую версию кодировки UTF-16 (включающую только 16-битные символы, но не суррогатные пары), что неверно. Однобайтовая кодировка в MS Windows (для русского языка используется кодовая страница Windows-1251) обозначается «ANSI».

Представление русского языка

В папке `labs-files/Файлы в формате простого текста - кодировки разные` представлены основные кодировки/таблицы для русского языка:

1. Многобайтовые кодировки универсальной кодовой таблицы Unicode:
   - — UTF-8 — суффикс имени файла utf8;
   - — UTF-16 — суффикс utf16;
   - — UTF-32 — суффикс utf32.
2. Однобайтовые расширения ASCII:
   - — КОИ-8 (код обмена информацией 8-битный) для русского алфавита (KOI8-R), использовавшаяся в России до широкого распространения MS DOS/MS Windows — суффикс koi8r;
   - — ISO 8859-5, разработанная ISO и IEC и одно время считавшаяся стандартной, но не использовавшаяся — суффикс iso;
   - — IBM CP866 (альтернативная кодировка ГОСТ), использовавшаяся в русифицированной MS DOS — суффикс dos;
   - — Windows-1251 (CP1251), использующаяся в русифицированной MS Windows — суффикс windows;
   - — MacCyrillic, использующаяся в Mac OS X — суффикс maccyrillic.

   Из-за совпадения кодов наиболее частотных русских букв утилиты распознавания кодировок регулярно путают Windows-1251 и MacCyrillic; для различения этих кодировок необходим дополнительный анализ.

Л1.3. Вопросы

1. Для чего нужен шестнадцатеричный редактор?
2. Какие функции libc используются для чтения/записи бинарных файлов?
3. Известно, что файл содержит осмысленный русскоязычный текст в одной из представленных в данной работе кодировок. Можно ли, используя только шестнадцатеричный редактор, без частотного анализа, отличить UTF-8, UTF-16, UTF-32: а) друг от друга, б) от однобайтовой кодировки? По каким признакам?

Лабораторная работа 2

Исследование статистических характеристик исходных текстов (как бинарных файлов, так и файлов в формате простого текста). Работа с кодовыми таблицами русского языка

Засчитывается только осенью 2025 г.; актуальная версия в https://gitlab.com/illinc/otik

Максимальная оценка ПИН обязательной части — 7 баллов. Максимум графы $7 + 7$ баллов включает бонусное задание Л2.№4.

Штраф за одно пропущенное обязательное задание -3 балла.

Л2.1. Задание на лабораторную работу

Задание Л2.№1. Разработайте программу или используйте набор программ (в частности, это может быть набор скриптов-однострочников, использующий стандартные утилиты GNU/Linux), который обрабатывает заданный файл Q.

Символом, как всегда, является *байт* (для x86/amd64 это 8 бит, *октет*), первичным алфавитом $A_1 = \{a_1, \ldots, a_{|A_1|}\}$ — множество возможных значений байта $\{\texttt{00}, \ldots, \texttt{FF}\}$ или, что то же самое, $\{0, \ldots, 255\}$.

Программа рассчитывает для Q:

— n — длину файла Q в символах первичного алфавита A_1 (в байтах, так как символ = 8-битный байт x86/amd64);

— $count(a_j)$ — общее количество вхождений каждого из символов $a_j \in A_1$ в Q (ненормированную целочисленную частоту $\nu_{\text{ненорм}}(a_j)$);

и оценивает, строя модель источника символов по файлу $Q = c_1 \ldots c_n$ в виде *источника без памяти* (модель для сжатия без учёта контекста):

— $p_{\text{БП}}(a_j) = \frac{count(a_j)}{n}$ — вероятность каждого из символов $a_j \in A_1$;

— $I_{\text{БП}}(a_j) = -\log_2 \Big(p_{\text{БП}}(a_j)\Big)$ [бит] — количество информации в каждом символе $a_j \in A_1$;

— $I_{\text{БП}}(Q) = \sum\limits_{i=1}^{n} I_{\text{БП}}(c_i)$ [бит], $c_i \in Q$ — суммарное количество информации в файле Q (не среднее на символ, которое в n раз меньше, а именно суммарное!).

Сравните:

а) длину $L(Q)$ [бит] $= n \cdot 8$ файла Q в битах и оценку $I_{\text{БП}}(Q)$ [бит] суммарного количества информации в Q в битах; для $I_{\text{БП}}(Q)$ [бит] печатать:
   — саму оценку $I_{\text{БП}}(Q)$ [бит] с двумя знаками после запятой;
   — её дробную часть $\{I_{\text{БП}}(Q)$ [бит]$\}$ в экспоненциальной форме.

б) длину $L(Q)$ [октетов] $= n$ файла Q в октетах (8-битных байтах x86/amd64) и оценку в октетах:
   — $I_{\text{БП}}(Q)$ [октетов] $= \frac{I_{\text{БП}}(Q)\text{ [бит]}}{8}$ (с двумя знаками после запятой);
   а также *оценки снизу* длин в октетах для сжатия без учёта контекста:
   — длины E [октетов] $= \lceil I_{\text{БП}}(Q)$ [октетов]$\rceil$ только сжатого текста, без информации, необходимой для его декодирования;
   — длины G_{64} [октетов] $= E + 256 \cdot 8$ архива, где к сжатому тексту добавляется таблица $(\nu(\texttt{00}), \ldots, \nu(\texttt{FF}))$ из 256 ненормированных 64-битных частот $\nu_{\text{ненорм}}(j) = count(j)$;
   — длины G_8 [октетов] $= E + 256 \cdot 1$ архива, где к сжатому тексту добавляется таблица из 256 нормированных 8-битных частот $\nu_{\text{норм}}(j)$.

Все полученные величины необходимо напечатать на экране или сохранить в виде текстового файла-отчёта; причём таблица характеристик символов алфавита $j \in A_1$ (символ, то есть байт j — в виде шестнадцатеричного значения байта, а не ASCII-символа; ненормированная частота $count(j)$; оценки вероятности $p(j)$ и количества информации $I(j)$) должна либо печататься дважды:
— отсортированной по алфавиту (по значению j);
— отсортированной по убыванию $count(j)$;
либо в программе необходимо предусмотреть пересортировку.

Проверьте разработанную программу на файлах различного формата (не только простом тексте; в том числе и на бинарных).

Выгодно ли применять к проанализированным файлам сжатие без учёта контекста? Нужно ли нормировать частоты?

Для побайтового чтения используйте $fread()$ или её аналог в используемом языке программирования.

Для проверки корректности используйте файлы с заранее известным $I_{\text{БП}}(Q)$: для четырёх разных произвольных октетов a, b, c, d верно $I_{\text{БП}}(a) = 0$ бит, $I_{\text{БП}}(ab) = 2$ бита, $I_{\text{БП}}(abcd) = 4 \cdot 2 = 8$ бит (1 байт x86) и т. п.

Задание Л2.№2. Бонус +3 балла для пар и НБ, обязательное для ПИН-троек. Разработайте программу, аналогичную Л2.№1, но считающую символом кодирования *печатный или управляющий символ Unicode,* а первичным алфавитом A_1 — множество символов Unicode (строчных букв, заглавных букв, цифр, различных пробельных символов, знаков препинания и т. п) в файле Q.

Обратите внимание, что Л2.№2 и связанное с ним Л2.№5 — *единственные* задания курса ОТИК, где символ кодирования не обязательно является байтом, а под исходным текстом не всегда понимается произвольный бинарный файл.

При оценивании длины архива учитывайте, что первичный алфавит A_1 Л2.№2, в отличие от Л2.№1, имеет переменные длину и состав. Соответственно, сохранять в архиве нужно не массив из 256 частот $\Big(\nu(\texttt{00}), ..., \nu(\texttt{FF})\Big)$, а:

— 64-битную длину $|A_1|$ алфавита;

— массив из $|A_1|$ пар символ-частота: $\Big(\big(a_1, \nu(a_1)\big), ..., \big(a_{|A_1|}, \nu(a_{|A_1|})\big)\Big)$, где либо в каждой паре символ $a_j \in A_1 \subseteq \text{Unicode}$ имеет переменную длину (UTF-8 или UTF-16) либо в каждой паре 32-битен (UTF-32).

Какой будет длина в октетах этих данных для исследуемых файлов?

Какой была бы длина в октетах, если бы сохранялись частоты не для символов файла $A_1 \subseteq \text{Unicode}$, а для всего Unicode (длина и состав алфавита постоянны, сохраняются только частоты $\Big(\nu(0), ..., \nu\big(|\text{Unicode}| - 1\big)\Big)$)?

Исследуйте один и тот же длинный текстовый файл в кодировке UTF-8 программами Л2.№1 и Л2.№2. Как выбор первичного алфавита влияет на:

— оценку $I_{\text{БП}}(Q)$ без учёта контекста;

— оценку снизу длин архива для сжатия без учёта контекста?

Повторите анализ для других длинных текстовых файлов в кодировке UTF-8.

Задание Л2.№3. Рассчитайте, используя программу Л2.№1, частоты октетов в файлах, являющихся простым текстом в различных кодировках (папка `labs-files/Файлы в формате простого текста - кодировки разные`). Исследуйте несколько разных осмысленных русскоязычных текстов и все представленные кодировки. Определите 4 наиболее частых октета среди всех используемых и 4 наиболее частых октета, не являющихся кодами печатных символов ASCII. Обратите внимание на распределение октетов многобайтовых кодировок.

Рассчитайте частоты октетов в файле, соответствующем варианту $(№ - 1)\%9$ в папке `labs-files/Варианты 2 - определение кодировки простого текста` (далее — файл W).

Определите, является ли W простым русскоязычным текстом в одной из стандартных кодировок (один из вариантов представляет собой нерусскоязычный текст); если да — определите кодировку.

Задание Л2.№4. Бонус $+7$ (добавляется к л/р, а не к общему бонусу) — источник с памятью.

Разработайте программу, которая по заданному файлу Q рассчитывает:

— $count(a_j a_k)$ — количество вхождений подстрок $a_j a_k$;

— $count(a_j *)$ — общее количество вхождений любых двухсимвольных подстрок, начинающихся с a_j (для всех символов, кроме последнего символа файла, $count(a_j *) = count(a_j)$;

и оценивает, строя модель источника символов по файлу $Q = c_1 \dots c_n$ в виде *стационарного источника Маркова первого порядка:*

— условную вероятность $p(a_k|a_j) = \frac{count(a_j a_k)}{\sum_k count(a_j a_k)} = \frac{count(a_j a_k)}{count(a_j *)}$ каждой пары символов $a_j, a_k \in A_1$;

— суммарное количество информации $I_{\text{CM1}}(Q)$ в файле Q в битах и байтах (безусловную вероятность $p(c_1 = a_j)$ считайте равной $\frac{1}{256}$);

аналогично Л2.№1 (символ кодирования = байт, как всегда).

Для проверки корректности используйте файлы с заранее известным $I_{\text{CM1}}(Q)$: $I_{\text{CM1}}(a) = I_{\text{CM1}}(ab) = I_{\text{CM1}}(abcd) = I_{\text{CM1}}(abab) = 8$ бит (1 байт x86); $I_{\text{CM1}}(abac) = 10$ бит (1,25 байта x86); $I_{\text{CM1}}(aabacad) = 16$ бит (2 байта x86).

Проверьте разработанную программу на файлах различного формата (не только простом тексте; в том числе и на бинарных). Сопоставьте результат с Л2.№1.

Таблицу частот какого размера нужно сохранить вместе со сжатым текстом длины $I_{\text{CM1}}(Q)$, чтобы успешно его декодировать? Выгодно ли такое сжатие?

Задание Л2.№5. Бонус +3 балла. Разработайте программу, аналогичную Л2.№4, но считающую символом кодирования *печатный или управляющий символ Unicode,* аналогично Л2.№2.

Проверьте разработанную программу, сопоставьте результат с Л2.№4 и Л2.№2.

Л2.2. О терминах

Символ кодирования в ТИ есть элемент (квант) качественной информации. В кодировании (то есть сжатии, защите от помех или шифровании) символом является, как правило, **байт** (для x86 — октет); изредка — отдельный бит (символ вторичного алфавита Хаффмана или первичного алфавита Хэмминга) или битовый блок отличной от байта длины; и никогда — печатный символ ASCII, KOI-8, Unicode etc (технически реализовать можно, как показывают задания Л2.№2/Л2.№5, но смысла в этом нет).

Текст, строка — последовательность символов в указанном выше смысле (в общем случае бинарный файл и его фрагменты). **Алфавит** — множество всех возможных символов в указанном выше смысле.

Не обманывайтесь принятой в кодировании терминологией!

Кто обманется и воспользуется функцией $fgetc()$ (или, тем более, $readLine()$, $split()$ и т. п.) где-то, кроме Л2.№2/Л2.№5 — минус 2 балла.

Л2.3. Об округлении

Если количество информации или оценка длины чего-либо округляется до целого числа битов/октетов, округление должно выполняться **вверх.** Для хранения десяти полных байтов и ещё одного бита понадобится одиннадцать байтов.

Л2.4. О единицах измерения

Единственная безразмерная величина в Л2 — вероятность p и её оценки.

Количество информации, как и длина файла, может быть измерено как в битах, так и в кратных единицах: байтах, килобайтах и др.; диадах, триадах, тетрадах, октетах и др. Значение n (длина файла в байтах) измеряется в байтах x86/amd64, то есть **октетах.** Количество информации, рассчитываемое по формуле $I = -\log_2 p$, измеряется в **битах** — в октеты рассчитанное значение либо необходимо переводить (I [октетов] $= \frac{I\text{ [бит]}}{8}$), либо изначально использовать формулу Шеннона в виде I [октетов] $= -\log_{256} p = -\frac{\log_2 p}{8}$.

Если кто-то станет сравнивать численное значение n [октетов] с численным значением I [бит] и делать из соотношения этих чисел какие-то выводы — Л2 не будет засчитана, пока команда не разберётся с используемыми единицами измерения.

Лабораторная работа 3

Проектирование формата файлов

Засчитывается только осенью 2025 г.; актуальная версия в https://gitlab.com/illinc/otik

Цель работы: научиться создавать и обрабатывать двоичные файлы на ЯВУ.

Максимальная оценка ПИН обязательной части — 7 баллов.
Штраф за одно пропущенное обязательное задание -2 балла.

Л3.1. Задание на лабораторную работу

Задание Л3.№1. Выберите сигнатуру для собственного формата файлов, которая будет использоваться во всех дальнейших работах (6 байт).

Разработайте программу-кодек, состоящую из двух частей — *кодера* и *декодера*.

1. *Кодер* по заданному файлу Q создаёт архив R формата, описанного в таблице Л3.1.

Нулевая версия формата архива

Таблица Л3.1

Смещение поля (байты)	Размер поля (байты)	Описание поля
0	6	Сигнатура формата (выбранное выше 6-байтовое значение)
6	2	Версия формата (беззнаковое 16-битное целое): для задания Л3.№1 — 0
8	8	Исходная длина n файла в байтах (беззнаковое 64-битное целое)
16	n	«Сырые» данные исходного файла (несжатый текст)

2. *Декодер* по заданному архиву R:
   - — проверяет сигнатуру на соответствие выбранной в Л3.№1 и версию формата;
   - — при корректных сигнатуре и версии восстанавливает файл $\widetilde{Q}$ (который должен совпадать с исходным Q) по данным архива R.

Кодер и декодер могут быть реализованы как в виде двух раздельных модулей, так в одной программе (в последнем случае действие задаётся ключом командной строки или выбором меню — то есть должна быть возможность раздельного вызова кодера и декодера, а не только слитного преобразования $Q \to R \to \widetilde{Q}$).

Проверьте при помощи шестнадцатеричного просмотровщика/редактора, что поля заголовка созданного архива соответствуют таблице Л3.1; при помощи утилиты GNU/Linux $diff$ проверьте побайтовое совпадение распакованного $\widetilde{Q}$ с исходным Q.

При выполнении работы в MS Windows аналог $diff$ можно найти, используя справочную команду консоли $help$.

Задание Л3.№2. Разработайте и опишите формат файла-архива для дальнейших работ (не обязан развивать таблицу Л3.1; формат Л3.№2 может отличаться от Л3.№1 размером сигнатуры, но её начало должно сохраниться). Архив обязательно содержит заголовок и закодированные данные.

Заголовок обязательно включает:

— сигнатуру (в дальнейшем все дальнейшие версии архивов команды должны использовать сигнатуру из Л3.№2);
— ненулевую версию формата (1 и выше; при разделении на мажорную и минорную — 0.1 и выше);
— коды использованных алгоритмов сжатия и защиты от помех (с учётом того, что сжатие без учёта контекста применяется поверх сжатия с его учётом — кодов алгоритма сжатия будет два);
— исходную длину файла в символах кодирования (то есть байтах);

а также любые необходимые, по мнению автора, поля.

Формат должен предусматривать возможность добавления служебных данных для различных алгоритмов сжатия и защиты от помех (например, массив частот для методов сжатия без учёта контекста) без кардинальной его переработки.

В качестве кодов алгоритмов, соответствующих отсутствию сжатия и защиты от помех, используйте значение 0.

Задание Л3.№3. Разработайте программу-кодек (аналогично Л3.№1), создающую архив формата, разработанного в задании Л3.№2.

Л3.2. О терминах

Символ кодирования = байт x86 = октет, а не печатный символ ASCII, KOI-8, Unicode etc.

Несжатый текст Q = любой бинарный файл; сжатый текст, который следует в R после кода алгоритма и служебной информации — тоже бинарный файл.

Не обманывайтесь принятой в кодировании терминологией! (Кто обманется и воспользуется функциями $fgetc()$ и т. п. — минус 2 балла).

«Маркером» текстового представления является не только $fgetc()$, но и — тем более — функции форматированного (что значит текстового) ввода-вывода и обработки строк, в частности, $readLine()$, $split()$ и т. п.

В большинстве библиотек для ввода-вывода «сырых» бинарных данных есть только две функции:
— аналог $fread()$ для чтения из файла;
— и аналог $fwrite()$ для записи;
и их *достаточно.*

Л3.3. Дополнительные бонусные и штрафные баллы

-2 балла за текстовое представление архива или если кодер обрабатывает только файлы в текстовом представлении — см. раздел Л3.2.

-2 балла, если чтение/запись в файл выполняется средствами не того ЯВУ, который исследован в $Л0^{НД}$.№1.

-1 балл, если структура архива, создаваемого кодеком в задании Л3.№3, не совпадает с описанной в задании Л3.№2.

-1 балл, если декодер при некорректной сигнатуре архива R всё равно создаёт файл $\widetilde{Q}$.

$+7$ баллов, если в структуре заголовка Л3.№2 и программе Л3.№3 предусмотрена возможность собрать в один архив и восстановить ещё и иерархическую структуру папок (не более $+4$, если можно собрать несколько файлов, но пути не сохраняются).

Лабораторная работа 4

Сжатие данных без учёта контекста (энтропийное сжатие)

Засчитывается только осенью 2025 г.; актуальная версия в https://gitlab.com/illinc/otik

Максимальная оценка ПИН обязательной части — 10 баллов. Максимум графы $30 = 10 + 20$ включает все бонусные задания и бонус за качество исполнения.

Для НБ — бонус $+16$ полная, $+8$ упрощённая.

Штраф за одно пропущенное обязательное задание -3 балла.

Л4.1. Упрощённое задание (не более 5 баллов за работу)

Используйте демонстрационные программы любой свободной библиотеки для сжатия методом Хаффмана, Шеннона—Фано, Шеннона либо целочисленным арифметическим (интервальным) методом. Узнайте из документации, какой метод используется и что является символом кодирования.

Продемонстрируйте работу кодера и декодера. Сопоставьте длину сжатых данных с оценкой количества информации в исходном файле (Л2.№1).

Отличие упрощённого задания от полного, реализованного с помощью библиотеки со свободной лицензией — в управлении заголовком.

В упрощённом задании студент использует демонстрационную программу «как есть»; полное может быть засчитано только в том случае, если запись/чтение заголовка реализованы студентом и могут быть изменены.

Засчитывается только в том случае, если студент может доказать, используя документацию библиотеки, что используется именно заявленный студентом метод сжатия без учёта контекста и только он (а не, например, связка LZW+Хаффман).

Л4.2. Задание на лабораторную работу

Задание Л4.№1. Разработайте программу-кодек (аналогично Л3.№3), реализующую сжатие/разжатие файла методом Хаффмана.

Сигнатура формата должна совпадать с выбранной в Л3.№1. Если в формат необходимо внести какие-либо изменения по сравнению с Л3.№2 — они должны быть описаны в отчёте, а номер версии — изменён.

Также в отчёте обязательно должны быть описаны:

— код алгоритма, не равный 0 (если выполняются бонусные задания ниже — разным алгоритмам должны соответствовать разные коды);
— состав информации для декодирования и формат её хранения в файле;
— принятые при построении дерева уточнения (порядок сортировки при совпадении частот и т. п.).

Сопоставьте длину сжатых данных с оценкой количества информации в исходном файле по стационарной модели без памяти (Л2.№1), а длину архива — с её оценкой снизу (Л2.№1).

Задание Л4.№2. Разработайте программу, рассчитывающую для файла коды Хаффмана (собственно архивы со сжатыми данными можно не создавать):

64) по ненормализованным частотам байтов $\nu_{64}(j) = count(j)$ в диапазоне $0 \dots 2^{64} - 1$ (для хранения одного значения необходимо $|\nu_{64}| = 8$ байт);
32) по усечённым (для файла длиной до $2^{32} - 1$) или нормализованным к диапазону $0 \dots 2^{32} - 1$ для больших файлов частотам ($|\nu_{32}| = 4$ байта);
8) по частотам, приведённым к диапазону $0 \dots 255$ ($|\nu_8| = 1$ байт);
4) по частотам, приведённым к диапазону $0 \dots 15$ ($|\nu_4| = \frac{1}{2}$ байта).

Частоты нормализовывать таким образом, чтобы ноль переходил в 0, единица в 1 (то есть чтобы ненулевые частоты не превращались в нулевые).

> В отличие от Л2.№1, где рассчитывается оценка снизу длины сжатого текста и длины архива — здесь рассчитывается точное значение.

Для каждого $|\nu_B|$ программа должна рассчитывать:

— длину E_B сжатых данных файла в байтах;
— общую длину G_B данных, необходимых для распаковки (сжатых данных E_B и массива частот) в байтах: $G_B = E_B + 256 \cdot \frac{B}{8} = E_B + 32 \cdot B$.

Для файла программа должна рассчитывать наиболее выгодную разрядность B^* частот $\left(G_{B^*} = \min_B(G_B)\right)$ для сжатия методом Хаффмана.

Для каждого из нескольких различных файлов сравните E_{64}, E_{32}, E_8 и E_4, а также G_{64}, G_{32}, G_8 и G_4; сравните B^*; запишите в отчёт.

Насколько выводы Л4.№2, учитывающие огрубление частот при приведении к заданному диапазону, отличаются от предварительных выводов Л2.№1?

Выгодно ли встраивать в реализацию алгоритма подбор B^* для конкретного файла? Учитывайте время и необходимость сохранения такого B^* в заголовке.

Какая фиксированная для алгоритма разрядность B^{**} лучше подходит для использования (удобнее в чтении/записи и при этом даёт достаточно малое G_B)? Предложите способ представления 256 B^{**}-битных частот в виде $32 \cdot B^{**}$ байтов.

+2 балла, если программа перебирает для файла все возможные разрядности B от 1 до 64 бит и рассчитывает B^* среди всех, а не только 64/32/8/4.

+2 балла за графическое изображение зависимостей $\frac{E_B}{E_{64}}$ и $\frac{G_B}{G_{64}}$ от B для набора из 10 или более файлов разного типа. В итоге должно получиться одно изображение для E (либо со множеством графиков на нём, либо с «ящиками с усами») и одно для G, независимо от количества файлов — если на каждый файл будет свой график, баллы не начисляются.

Графики и сделанные из них выводы — продублировать лектору, для истории.

Задание Л4.№3. Разработайте универсальный декодер разработанного формата, который:

— проверяет сигнатуру на соответствие выбранной в Л3.№1, при некорректной сигнатуре прекращает работу;
— при корректной сигнатуре — проверяет номер версии формата: для предыдущих версий формата вызывает свою старую версию, при ещё неизвестном номере версии прекращает работу;
— при корректных сигнатуре и версии — коды алгоритмов, после чего вызывает соответствующий им декодер из заданий Л3.№3 или Л4.№1.

В дальнейшем необходимо будет дополнять анализ при добавлении нового алгоритма (при этом любая модификация — например, реализация метода Хаффмана с другим порядком сортировки при совпадении частот — должна оформляться именно как новый алгоритм с новым кодом).

Задание Л4.№4. Бонус +3 балла.

Разработайте «интеллектуальный» кодер, который анализирует суммарный объём n_{compr} сжатых данных и информации для декодирования Л4.№1 и, если $n_{compr} \geqslant n$, записывает в полученный архив несжатый текст исходного файла (то есть использует вместо кодера Л4.№1 кодер Л3.№3; код алгоритма при этом также должен быть 0).

Сразу предусмотрите флаг для отключения анализа и принудительного использования заданного алгоритма.

Задание Л4.№5. Бонус $+10$ (добавляется к л/р, а не к общему бонусу) — арифметический/интервальный кодек.

Разработайте программу-кодек (аналогично Л4.№1–Л4.№4), реализующую сжатие/разжатие файла целочисленным арифметическим (интервальным) методом.

Задание Л4.№6. Бонус +3 балла для пар, обязательное для троек.

Разработайте программу-кодек (аналогично Л4.№1–Л4.№4), реализующую сжатие/разжатие файла методом, соответствующим варианту (таблица Л4.1).

Варианты Л4.№6

Таблица Л4.1

$(№ - 1)\%2 + 1$	**Вариант**
1	метод Шеннона
2	метод Шеннона—Фано

Л4.3. Дополнительные бонусные и штрафные баллы

-4 балла за текстовое представление архива или если кодер обрабатывает только файлы в текстовом представлении — см. раздел Л3.2.

-2 балла, если массив частот может для какого-то одного файла иметь длину более 256 байтов.

-2 балла, если формат архива не соответствует Л3.№2 (штраф не начисляется, если в отчёте описана новая версия формата, которая логичнее исходной).

-4 балла, если даже сигнатура не совпадает с Л3.№1.

-2 балла, если ранее разработанный декодер Л3.№3 пытается декодировать архив Л4.№1, а не выдаёт сообщение о несоответствии алгоритма.

$+1$ балл, если при работе с несколькими файлами/папками каждый из файлов имеет собственный код алгоритма и информацию для декодирования (а в Л4.№4 — анализируется отдельно).

$+4$ балла, если архив позволяет как включить для каждого файла собственный код алгоритма и информацию для декодирования, так и рассмотреть совокупность файлов как единый исходный текст (но при декодировании восстановить исходную структуру файлов).

Лабораторная работа 5

Сжатие данных с учётом контекста

Засчитывается только осенью 2025 г.; актуальная версия в https://gitlab.com/illinc/otik

Максимальная оценка ПИН обязательной части — 10 баллов. Максимум графы $30 = 10 + 20$ включает все бонусные задания и бонус за качество исполнения.

Для НБ — бонус $+16$ полная, $+8$ упрощённая.

Штраф за одно пропущенное обязательное задание -3 балла.

Л5.1. Упрощённое задание (не более 5 баллов за работу)

Разработайте кодер и декодер «наивного» RLE. Продемонстрируйте работу кодера и декодера.

Штраф −3 балла за текстовое представление архива или если кодер обрабатывает только файлы в текстовом представлении — см. раздел Л3.2.

Засчитывается только в том случае, если все студенты команды знают:

а) сколько байтов они отводят на код $\widetilde{L}$ длины цепочки L;

б) в каком виде записывают длину цепочки $L \geqslant 1$ в эти байты: $\widetilde{L} = L - 1$ или $\widetilde{L} = L$;

в) какое максимально допустимое значение $L_{\max}$ для длины цепочки L следует из сделанного ими выбора (а) и (б);

г) в каком порядке записывают код цепочки «наивного» RLE: $(\widetilde{L},\ c)$ или $(c,\ \widetilde{L})$;

и если они корректно обрабатывают бóльшее, чем $L_{\max}$, количество повторений подряд одного символа c в файле.

Л5.2. Задание на лабораторную работу

Задание Л5.№1. Разработайте программу-кодек, реализующую сжатие/разжатие файла методом RLE согласно варианту (таблица Л5.1). Здесь и далее в разных вариантах используются разные коды/алгоритмы, реализующие один заданный метод — семейство кодов и алгоритмов. Если вы не поняли, какой код нужно реализовать в вашем варианте — спрашивайте. «Какая-нибудь» реализация заданного метода, не соответствующая варианту — не засчитывается.

Вычислите минимальные и максимальные возможные значения L для своего варианта (для всех возможных случаев).

Сопоставьте длину сжатых данных с оценкой количества информации в исходном файле по модели Маркова (Л2.№4), а также с объёмом таблицы частот в Л2.№4.

Задание Л5.№2. Разработайте программу-кодек, реализующую сжатие/разжатие файла методом LZ78 согласно варианту (таблица Л5.3).

Задание Л5.№3. Бонус $+10$ (добавляется к л/р, а не к общему бонусу). Разработайте программу-кодек, реализующую сжатие/разжатие файла методом LZ77 согласно варианту (таблица Л5.3).

Варианты кодов семейства RLE

Таблица Л5.1

$(№-1)\%3+1$	**Вариант**
1	флаг-бит сжатая/несжатая цепочка; цепочка из $L \geqslant 3$ одинаковых символов $c...c$ как $\Big(1:1,\ (L-3):7,\ c:8\Big)$, из $L \geqslant 1$ разных $c_1...c_L$ — как $\Big(0:1,\ (L-1):7,\ c_1:8,\ ...\ c_L:8\Big)$
2	префикс p; цепочка из $L \geqslant 4$ одинаковых символов $c...c,\ c \neq p$ как $\Big(p:8,\ (L-3):8,\ c:8\Big)$, цепочка из $L \geqslant 2$ символов $p...p$ как $\Big(p:8,\ (L-1):8,\ p:8\Big)$, одиночный p как $(p:8,\ 0:8)$
3	префикс p; цепочка из $L \geqslant 4$ одинаковых символов $c...c,\ c \neq p$ как $\Big(p:8,\ (L-4):8,\ c:8\Big)$, цепочка из $L \geqslant 1$ символов $p...p$ как $\Big(p:8,\ (L-1):8,\ p:8\Big)$, в том числе одиночный p как $(p:8,\ 0:8,\ p:8)$

Варианты кодов семейства LZ78

Таблица Л5.2

$(№-1)\%2+1$	**Вариант**
1	концепт Зива и Лемпеля 1978 года
2	LZW

Запись в таблице $(S:10,\ L:6)$ обозначает: сначала записывается 10 бит S, затем 6 бит L. Для концепта Зива и Лемпеля символ c записывается как $(c:8)$ (то есть просто как байт c); для реализации с односимвольным префиксом p символ $c \neq p$ также записывается как $(c:8)$. Если ссылка отделяется от несжатого текста односимвольным префиксом p, значение p выбирать для каждого исходного текста Q отдельно, исходя из частот символов в Q.

Вычислите минимальные и максимальные возможные значения S и L для своего варианта.

Задание Л5.№4. Бонус +2 балла для пар, обязательное для троек. Доработайте универсальный декодер из задания Л4.№3 и кодер из задания Л4.№4 с учётом всех реализованных алгоритмов.

Варианты кодов семейства LZ77

Таблица Л5.3

$(№-1)\%7+1$	**Вариант**
1	концепт Зива и Лемпеля; ссылка $\{S, L\}$ как $(S:10,\ \ L:6)$
2	концепт Зива и Лемпеля; ссылка $\{S, L\}$ как $(L:6,\ \ S:10)$
3	флаг-бит ссылка/цепочка; ссылка $\{S, L\}$ как $\Big(1:1,\ \ (L-3):7,\ \ (S-1):8\Big)$, цепочка $c_1 ... c_L$ как $\Big(0:1,\ \ (L-1):7,\ \ c_1:8,\ \ ...\ c_L:8\Big)$,
4	флаг-биты ссылка/символ группируются во флаг-байты; ссылка $\{S, L\}$ записывается как $\Big(S:10,\ \ (L-3):6\Big)$ — LZJB
5	флаг-биты ссылка/символ группируются во флаг-байты; ссылка $\{S, L\}$ записывается как $\Big((L-3):6,\ \ S:10\Big)$
6	префикс p; ссылка $\{S, L\}$ как $\Big(p:8,\ \ S:10,\ \ (L-4):6\Big)$, символ $c=p$ в тексте — как $\Big(p:8,\ \ 0:8,\ \ 0:8\Big)$
7	префикс p; ссылка $\{S, L\}$ как $\Big(p:8,\ \ (L-3):6,\ \ (S-1):10\Big)$, символ $c=p$ в тексте — как $\Big(p:8,\ \ 0:8\Big)$

Задание Л5.№5. Бонус +5 для пар, +2 для троек, обязательное для четвёрок. Доработайте универсальные кодер и декодер так, что к одному файлу было возможно применить последовательно:

1) вначале сжатие с учётом контекста;
2) затем сжатие без учёта контекста.

Баллы не начисляются, если кодер/декодер сводятся к последовательному запуску двух программ, то есть если на первом проходе формируется заголовок формата Л3.№2, и второй проход рассматривает этот заголовок как часть исходного текста.

Конкретные алгоритмы выбираются при запуске: сжатие с контекстом из реализованных в данной л/р, без контекста — из реализованных в предыдущей л/р.

Лабораторная работа 6

Помехозащитное кодирование

Засчитывается только осенью 2025 г.; актуальная версия в https://gitlab.com/illinc/otik

Исключена после переноса ОТИК с четвёртого года обучения на третий. Если кто-то выполнит — максимальная оценка обязательной части 10 баллов; оценка добавляется к сумме бонусов.

Штраф за одно пропущенное обязательное задание -2 балла.

Л6.1. Упрощённое задание (не более 7 баллов за работу)

Используйте демонстрационные программы любой свободной библиотеки для защиты от помех методом Хэмминга либо Рида—Соломона. Узнайте из документации, какой метод используется, какое количество ошибок E он исправляет в блоке, размер блока, тип исправляемой ошибки.

Продемонстрируйте, что E ошибок при декодировании исправляется, а $E+1$ — уже нет.

Л6.2. Задание на лабораторную работу

Задание Л6.№1. Разработайте программу-кодек, реализующую защиту данных от помех методом Хэмминга, позволяющую исправить одиночную инверсию в блоке и обнаружить двойную инверсию в блоке.

Размер блока до кодирования соответствует варианту (таблица Л6.1). Код

Варианты размера блока до кодирования

Таблица Л6.1

$(№-1)\%2+1$	**Вариант**
1	$N=7$ байт
2	$N=8$ байт

должен быть систематическим, количество контрольных символов — целым (если в контрольном символе при заданном размере блока остаются неиспользуемые биты — продублируйте бит общей чётности).

Внесите в закодированные данные ошибки; убедитесь, что при декодировании действительно исправляется одиночная инверсия в блоке и обнаруживается двойная.

Задание Л6.№2. Разработайте программу-кодек, реализующую защиту данных от помех методом Рида—Соломона (используйте библиотеки, распространяемые по свободным лицензиям). Какова выбранная вами длина блока до коди-

рования (сколько информационных символов)? Сколько контрольных символов добавляется? Какое максимальное количество ошибок в блоке может быть исправлено?

Внесите в закодированные данные ошибки; убедитесь, что при декодировании действительно исправляется указанное выше количество ошибок.

Задание Л6.№3. К ключевым полям заголовка файла (сигнатура, версия, алгоритмы и т. п.) нельзя применить какой-либо помехозащитный код без нарушения читаемости заголовка, поэтому они должны обрабатываться отдельно (например, троироваться, или дублироваться с применением к дубликату защиты от помех, или добавленные контрольные символы заголовка должны быть помещены вне заголовка).

Добавьте новую версию формата, где заголовок защищён от помех даже в том случае, когда к данным защита не применяется.

Задание Л6.№4. Бонус +3 балла для пар, обязательное для троек. Доработайте универсальный декодер из задания Л4.№3 и кодер из задания Л4.№4 с учётом реализованных алгоритмов.

Задание Л6.№5. Бонус +7 для пар, +3 для троек, обязательное для четвёрок. Доработайте универсальные кодер и декодер так, что к одному файлу было возможно применить последовательно:

1) вначале сжатие с учётом контекста;
2) затем сжатие без учёта контекста;
3) в конце защиту от помех.

Баллы не начисляются, если кодер/декодер сводятся к последовательному запуску двух программ, то есть если на первом проходе формируется заголовок формата Л3.№2, и второй проход рассматривает этот заголовок как часть исходного текста.

Дополнительно +7 баллов, если кодер и декодер включают шифрование:

1) вначале сжатие с учётом контекста;
2) затем сжатие без учёта контекста;
3) после всех алгоритмов сжатия — шифрование;
4) в конце — защиту от помех.

Л6.3. Дополнительные бонусные и штрафные баллы

−2 балла, если размер блока Хэмминга — весь файл.

−2 балла, если код несистематический.

−2 балла, если в блоке после кодирования есть неиспользуемый бит, а двойная ошибка в блоке не распознаётся.

Приложение B. Коды ASCII

ASCII CONTROL CODE CHART

BITS b4 b3 b2 b1	b7 b6 b5 = 0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		SYMBOLS NUMBERS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0	16 DLE 20	32 SP 40	48 0 60	64 @ 100	80 P 120	96 ‘ 140	112 p 160
0 0 0 1	1 SOH 1	17 DC1 21	33 ! 41	49 1 61	65 A 101	81 Q 121	97 a 141	113 q 161
0 0 1 0	2 STX 2	18 DC2 22	34 ” 42	50 2 62	66 B 102	82 R 122	98 b 142	114 r 162
0 0 1 1	3 ETX 3	19 DC3 23	35 # 43	51 3 63	67 C 103	83 S 123	99 c 143	115 s 163
0 1 0 0	4 EOT 4	20 DC4 24	36 $ 44	52 4 64	68 D 104	84 T 124	100 d 144	116 t 164
0 1 0 1	5 ENQ 5	21 NAK 25	37 % 45	53 5 65	69 E 105	85 U 125	101 e 145	117 u 165
0 1 1 0	6 ACK 6	22 SYN 26	38 & 46	54 6 66	70 F 106	86 V 126	102 f 146	118 v 166
0 1 1 1	7 BEL 7	23 ETB 27	39 ’ 47	55 7 67	71 G 107	87 W 127	103 g 147	119 w 167
1 0 0 0	8 BS 10	24 CAN 30	40 (50	56 8 70	72 H 110	88 X 130	104 h 150	120 x 170
1 0 0 1	9 HT 11	25 EM 31	41) 51	57 9 71	73 I 111	89 Y 131	105 i 151	121 y 171
1 0 1 0	10 LF 12	26 SUB 32	42 * 52	58 : 72	74 J 112	90 Z 132	106 j 152	122 z 172
1 0 1 1	11 VT 13	27 ESC 33	43 + 53	59 ; 73	75 K 113	91 [133	107 k 153	123 { 173
1 1 0 0	12 FF 14	28 FS 34	44 , 54	60 < 74	76 L 114	92 \ 134	108 l 154	124 | 174
1 1 0 1	13 CR 15	29 GS 35	45 – 55	61 = 75	77 M 115	93] 135	109 m 155	125 } 175
1 1 1 0	14 SO 16	30 RS 36	46 . 56	62 > 76	78 N 116	94 ^ 136	110 n 156	126 ~ 176
1 1 1 1	15 SI 17	31 US 37	47 / 57	63 ? 77	79 O 117	95 ¨ 137	111 o 157	127 DEL 177

LEGEND: dec / CHAR / hex oct

Victor Eijkhout
Dept. of Comp. Sci.
University of Tennessee
Knoxville TN 37996, USA

Оглавление