СЖАТИЕ ДАННЫХ. АЛГОРИТМ ХАФФМАНА

Аннотация

Статья посвящена алгоритмам сжатия данных. Приводится краткий обзор методов сжатия данных с потерями и без потерь информации. Подробно описан алгоритм сжатия данных Хаффмана, с использованием терминов из теории графов. Рассмотрен пример работы алгоритма для кодирования фразы «Veni, vidi, vici» на основе бинарного дерева. Показаны эффективность и недостатки данного метода сжатия данных. Статья снабжена иллюстрациями.

Ключевые слова

сжатие данных, алгоритм Хаффмана, бинарные деревья, кодирование

АВТОРЫ

Виноградова Марина Станиславовна,
кандидат физико-математических наук, доцент
ФГБОУ ВО «Московский государственный технический университет им. Н. Э. Баумана», г. Москва,
m-s-vinogradova@yandex.ru

Ткачева Ольга Сергеевна,
младший научный сотрудник,
Институт проблем управления им. В. А. Трапезникова РАН, г. Москва,
tkolga17@gmail.com

Введение

Вопрос уменьшения объема, занимаемого данными, хранящимися в базе данных без потери или с минимальной потерей информации остро встал, практически, одновременно с появлением систем для хранения данных. Данные, как правило, хранятся в виде, позволяющем их наиболее простое использование, каждый символ кодируется кодом фиксированной длины, например, в ASCII коде каждый символ занимает 8 бит. Такое представление данных требует значительные объемы памяти для их хранения. Поэтому сжатие данных является одним из наиболее актуальных направлений современных технологий хранения данных. Сжатием данных обычно называют такое преобразование данных, которое позволяет уменьшить занимаемый ими объём. Сжатие данных позволяет как уменьшить физический размер базы данных, так и повысить производительность работы СУБДБ, и снизить требования к объему оперативной памяти. Скорость выполнения запросов в базах данных увеличивается, поскольку операции ввода-вывода информации компьютером (считывание информации с диска и запись на диск) являются достаточно затратными по времени, соответственно, чем больше объем данных, тем больше времени требуется на считывание информацию с диска или записи ее на диск [1,2,3].

Несмотря на то, что относительная стоимость устройств хранения информации значительно уменьшилась, а объем памяти носителей увеличился за последние десятилетия, вопрос уменьшения объема, занимаемого данными, по-прежнему актуален.

Связано это с тем, что одновременно с развитием технических средств растут и объемы хранимой информации.

Еще можно отметить, что сжатие данных по сути является их кодированием и, следовательно, способствует криптографической защите информации [1,2,3].

Методология и результаты исследования

Сжатие данных – это процесс преобразования данных, обеспечивающий уменьшение объема данных путем сокращения их избыточности. Целью сжатия данных является обеспечение уменьшения объема занимаемой данными памяти и увеличения скорости обработки.

Методы сжатия сжатии данных являются частным случаем методов кодирования информации. Напомним, что кодирование информации – это преобразование из одного вида представления информации к другому, удобному для хранения, передачи или обработки, результатом процесса кодирования является последовательность кодов [1]. Кодом называют взаимно-однозначное отображение конечного упорядоченного множества символов, принадлежащих некоторому конечному алфавиту, на другое, не обязательно упорядоченное, возможно более обширное множество символов для кодирования передачи, хранения или преобразования информации [4]. Декодирование – это преобразование кода символа в его исходный вид, т.е. обратное преобразование восстановления информации из закодированного представления [4].

Методы сжатия данных с точки зрения возможности полного восстановления данных, могут быть разделены на два больших класса: обратимое и необратимое сжатие.

1. Обратимое сжатие часто называют полностью обратимым или сжатием без потерь. Такой метод сжатия позволяет снизить объем данных без потери информации, т. е. позволяет восстановить закодированную (сжатую) порцию данных полностью без изменений. Обратимое сжатие используется в базах данных при кодировании текстовой информации.

2. Необратимое сжатие называют сжатием с потерями информации, это такой метод сжатия, при котором невозможно восстановить данные источника в полностью первоначальном виде. При использовании метода сжатия с потерями обеспечивается максимальная степень сжатия исходных данных, при этом часть данных отбрасывается. Такой метод сжатия обычно используется при кодировании графической, звуковой и видео информации.

Методы сжатия сжатии данных сточки зрения возможности использования их для сжатия информации любого типа можно разделить на универсальные и специальные. Универсальные методы предназначены для сжатия информации, содержащейся в файлах любого формата, поскольку рассматривают файл, как простую последовательность битов. Универсальные методы позволяют полностью восстанавливают исходную информацию. Они относятся к классу обратимых методов.

Специальные методы предназначены для сжатия информации определенного типа (текста, изображений, звука или видео). Такие методы учитывают специфику восприятия изображений, звука или видео человеком. При сжатии специальными методами можно добиться очень большой степени сжатия при потере качества за счет удаляется маловажной информации. Однако специальные методы не позволяют полностью восстановить исходную информацию и относятся к классу необратимых методов [1,2,3].

Далее будем универсальные обратимые методы сжатия информации.

Здесь можно выделить три основных группы методов сжатия.

1. Методы преобразования блока. В этой группе методов все данные разбиваются на блоки, и далее, каждый блок данных обрабатывается отдельно. Некоторые методы в этой группе, могут не приводить к заметному уменьшению объема данных, особенно ме-

тоды, основанные на перестановке блоков. Однако, следует отметить, что такая процедура может применятся в качестве предварительной обработки данных при сжатии. Структура данных в этом случаю улучшается, и применение в дальнейшем других методов сжатие проходит более эффективно. К блочным методам сжатия можно отнести статические варианты метода Шеннона-Фано и арифметического кодирования [3].

2. Преобразование потока. В этой группе методов не используются статистические методы, кодирование символов осуществляется только на основе тех данных, которые уже были обработаны. Входной поток данных анализируется и, если некоторая подстрока в потоке уже встречалась, то есть известна кодировщику, то все ее дальнейшие вхождения заменяются ссылками на первое вхождение такой подстроки. Примерами метода преобразования потока могут служить методы, основанные на алгоритме Лемпеля–Зива (Lempel-Ziv), так называемые, LZ – методы. Первоначальная версия метода была создана Лемпелем и Зивом в 1977 году (LZ77), доработана ими же в 1978 году ((LZ78)) и затем усовершенствована Уэлчем (Welch) в 1984 году, метод LZW (Lempel-Ziv-Welch) [3].

3. Статистические методы сжатия.

Статистические методы сжатия данных основаны на следующем принципе: наиболее часто встречающиеся элементы закодированы более короткими кодами, а реже встречающиеся – более длинными. В этом случае для хранения всех данных требуется меньше места, чем если бы все элементы представлялись кодами одинаковой длины.

Статистические методы можно разделить на две группы. методов.

1. Адаптивные или поточные методы. Кодирование производится с использованием вероятностей для новых данных, вычисленных на основании статистики по уже обработанным данным. В качестве примера адаптивных методов можно указать группу PPM-методов для потоков «слов», адаптивный вариант методов арифметического кодирования. Следует отметить, что в отличие от случая LZ–методов, ранее собранная статистика имеет тот же вес, что и недавняя. К недостаткам методов можно отнести и то, что считаются вероятными все возможные комбинации «слов», даже те, которые не встречались в потоке ранее и вероятнее всего никогда не встретятся [3]. Под «словом» в данном контексте понимается последовательность символов во входном алфавите, часто имеющая некоторый смысл.

2. Блочные методы. В этой группе методов отдельно кодируется статистика блока и добавляется к сжатому блоку. Примеры блочных методов: статические варианты методов Хаффмана, Шеннона–Фано, арифметического кодирования – для потоков "элементов [3].

Оптимальную длину кода можно определить через частоту (вероятность) появления элемента с использованием понятия *энтропия символа*.

Энтропией символа $\boldsymbol{c}$, имеющего вероятность появления $\boldsymbol{p_c}$, называют количество информации $\boldsymbol{H_c}$, содержащейся в $\boldsymbol{c}$, $\boldsymbol{H_c = p_c \cdot \log_2 p_c}$ [1].

Согласно теореме Шеннона о кодировании источника шифрования (Shannon's source coding theorem) элемент $\boldsymbol{c}$ вероятность появления которого равняется $\boldsymbol{p_c}$, выгоднее всего представлять кодом длиной $\boldsymbol{l_c = \log_2 p_c}$ битам [3].

Основные характеристики, использующиеся для оценки эффективности метода сжатия, это коэффициент сжатия $\boldsymbol{K_{compress}}$ и скорость сжатия $\boldsymbol{R}$ [1,3].

1. Коэффициент сжатия $\boldsymbol{K_{compress} = \frac{S_{output}}{S_{input}}}$, *где* $\boldsymbol{S_{output}}$ *– размер выходного файла (сжатые данные),* $\boldsymbol{S_{input}}$ *– размер входного файла (данные источника)*. При $\boldsymbol{0 < K_{compress} < 1}$ сжатие считается успешным.

2. Скорость сжатия R, единицей измерения скорости сжатия является «количество кодовых бит, приходящихся на отсчет данных источника» $R = \frac{k}{n}$, здесь $k =$

$\log_2|V|$ – размер сжатых данных в битах, *где* $|V|$ – мощность алфавита данных V, n – размер данных источника в битах.

Алгоритм Хаффмана

Одним из основных методов, используемыми для сжатия данных в базах данных в настоящее время является кодирование Хаффмана на основе кодового бинарного дерева.

Код Хаффмана является неравномерным и префиксным. Неравномерность кода означает, что символы имеют разную длину кодового слова (размер кода), коды символов, которые чаще встречаются в тексте, имеют меньший размер, а коды редко встречающихся символов, имеют больший размер. Префиксный называется такая кодировка, в которой ни один код не является началом другого кода, таким образом выполняется условие Фано и достигается однозначность при декодировании.

Поскольку в алгоритме Хаффмана строится кодовое бинарное дерево, перед его описанием напомним некоторые термины и определения из теории графов следуя [5].

Ориентированный граф $\boldsymbol{G}$ задается двумя множествами $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, где $\boldsymbol{V}$ – конечное множество, элементы которого называют *вершинами* или *узлами*; $\boldsymbol{E}$ – множество упорядоченных пар на $\boldsymbol{V}$, т.е. подмножество множества $\boldsymbol{V} \times \boldsymbol{V}$, элементы которого называют *дугами*. Дуга из вершины $\boldsymbol{u}$ в вершину $\boldsymbol{v}$ обозначают $(\boldsymbol{u}, \boldsymbol{v})$, или $\boldsymbol{u} \rightarrow \boldsymbol{v}$; $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{V} \times \boldsymbol{V}$; $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V}$.

Дугу $(\boldsymbol{u}, \boldsymbol{v})$ называют *заходящей* в вершину $\boldsymbol{v}$ и *исходящей* из вершины $\boldsymbol{u}$.

Полустепенью захода вершины $\boldsymbol{v}$ называют число $\boldsymbol{dg}^-(\boldsymbol{v})$ заходящих в нее дуг. *Полустепенью исхода* вершины $\boldsymbol{v}$ число $\boldsymbol{dg}^+(\boldsymbol{v})$ исходящих из нее дуг. Степенью $\boldsymbol{dg}(\boldsymbol{v})$ вершины $\boldsymbol{v}$ называют сумму полустепеней захода и исхода.

Путь в ориентированном графе $\boldsymbol{G}$ – это последовательность вершин (конечная или бесконечная) $\boldsymbol{v_0}, \boldsymbol{v_1}, \cdots, \boldsymbol{v_n}, \cdots$, такая, что $\boldsymbol{v_i} \longrightarrow \boldsymbol{v_{i+1}}\ \forall\ \boldsymbol{i}$, если существует $\boldsymbol{v_{i+1}}$. Для конечного пути $\boldsymbol{v_0}, \boldsymbol{v_1}, \cdots, \boldsymbol{v_n}$ число $\boldsymbol{n}$ называют *длиной* пути $(\boldsymbol{n} \geq \boldsymbol{0})$, то есть длина пути есть число его дуг, т.е. всех дуг, которые ведут из вершины $\boldsymbol{v_i}$ в вершину $\boldsymbol{v_{i+1}}$.

Простой путь – это путь, все вершины которого, кроме, быть может, первой и последней, попарно различны.

Простой путь ненулевой длины, начало и конец которого совпадают, называют *контуром*.

Ориентированный граф называют *связным*, если для любых двух его вершин $\boldsymbol{u}$, $\boldsymbol{v}$ вершина $\boldsymbol{v}$ достижима из вершины $\boldsymbol{u}$ или вершина $\boldsymbol{u}$ достижима из вершины $\boldsymbol{v}$ ($\boldsymbol{u} \Longrightarrow \boldsymbol{v}$ или $\boldsymbol{v} \Longrightarrow \boldsymbol{u}$).

Компонента связности (компонента) ориентированного графа – это максимальный связный подграф.

Ориентированный граф называют *сильно связным*, если для любых двух его вершин $\boldsymbol{u}$ и $\boldsymbol{v}$ вершина $\boldsymbol{v}$ достижима из вершины $\boldsymbol{u}$ и вершина $\boldsymbol{u}$ достижима из вершины $\boldsymbol{v}$ ($\boldsymbol{u} \Longrightarrow \boldsymbol{v}$ и $\boldsymbol{v} \Longrightarrow \boldsymbol{u}$).

Бикомпонента ориентированного графа – это его максимальный сильно связный подграф.

Неориентированный граф $\boldsymbol{G_1} = (\boldsymbol{V_1}, \boldsymbol{E_1})$ называют *ассоциированным* с ориентированном графом $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, если его множество вершин $\boldsymbol{V_1}$ совпадает с множеством вершин $\boldsymbol{V}$ ориентированного графа $\boldsymbol{G}$, а пара $(\boldsymbol{u}, \boldsymbol{v})$ образует ребро тогда и только тогда, когда $\boldsymbol{u} \neq \boldsymbol{v}$ и из $\boldsymbol{u}$ в $\boldsymbol{v}$ или из $\boldsymbol{v}$ в $\boldsymbol{u}$ ведет дуга, ($\boldsymbol{V_1} = \boldsymbol{V}$ и $\boldsymbol{E_1} = \{\{\boldsymbol{u}, \boldsymbol{v}\}: (\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$ или $(\boldsymbol{v}, \boldsymbol{u}) \in \boldsymbol{E}, \}$).

Ориентированный граф называют *слабо связным*, если ассоциированный с ним неориентированный граф связный. *Компонентой слабой связности* (*слабой компонентой*) ориентированного графа называют его максимальный слабо связный подграф.

Ориентированным деревом называют *бесконтурный* ориентированный граф, у которого *полустепень захода* любой вершины не больше *1* и существует ровно одна

вершина, называемая *корнем ориентированного дерева*, *полустепень захода* которой равна *0*.

Вершину v ориентированного дерева называют *потомком* (*подлинным потомком*) вершины u, если существует путь из u в v (путь ненулевой длины из u в v). В этом же случае вершину u называют *предком* (*подлинным предком*) вершины v, а если длина пути из u в v равна *1*, то вершину v называют *сыном* вершины u, которая при этом именуется *отцом* вершины v. Вершину, не имеющую потомков, называют *листом*.

Высота ориентированного дерева – это наибольшая длина пути из корня в лист.

Ориентированное дерево называют *бинарным*, если полустепень исхода любой его вершины не больше 2.

Бинарное ориентированное дерево называют *полным* (ПБОД), если из любой его вершины, не являющейся листом, исходят ровно две дуги, а уровни всех листьев совпадают. Если каждая *слабая компонента* ориентированного графа является ориентированным деревом, то такой граф называют *ориентированным лесом*.

Взвешенным (или *размеченным*) ориентированным графом называют пару $W = (G, \varphi)$, где $G = (V, E)$ – обычный ориентированный граф, φ – *весовая функция* (или *функция разметки*), то есть каждой дуге графа сопоставлено некоторое число φ, называемое *весом дуги*.

Метод сжатия информации на основе двоичных кодирующих деревьев был предложен Д. А. Хаффманом в 1952.

Идея алгоритма состоит в том, чтобы наиболее часто встречающиеся символы имели более короткие коды, символы, встречающиеся реже всего, имели очень длинный код. Алгоритм сжатия данных Хаффмана, как канонический, так и его адаптивные версии, обладают достаточно высокой эффективностью и лежат в основе многих других методов, используемых в алгоритмах сжатия данных.

Каждому символу в кодируемом тексте присваивается вес, равный его частотой его появления (или его вероятности). Под символом обычно понимают некий повторяющийся элемент исходной строки, как печатный знак, так и любую битовою последовательность, под кодом или кодовым словом – кодирующую последовательность битов (не ASCII или какой-ибо другой исходный код символа).

Сжатие на основе алгоритма Хаффмана состоит из двух этапов. Сначала на подготовительном этапе читаются все входные данные, подсчитываются частоты встречаемости или вероятности всех символов, массив символов сортируется в соответствии с рассчитанными значениями частот. Затем, на втором этапе по этим данным строится дерево Хаффмана, по которому вычисляются коды символов.

Рассмотрим алгоритм подробнее.

Алгоритм построения бинарного ориентированного кодового дерева Хаффмана

1. Символы входного алфавита образуют массив входных символов. Каждому символу сопоставляется вес, равный частоте или вероятности появления этого символа.

2. Полученный массив символов сортируется по возрастанию весов символов. Формируется очередь, каждый элемент которой интерпретируется как изолированная вершину графа, помеченная символом входного алфавита, вершине приписывается вершине вес, соответствующий этому входному символу. Каждую изолированную вершину рассматривается как корень бинарного ориентированного дерева нулевой высоты, а все множество вершин – как лес.

3. Из головы очереди последовательно выбираются два элемента, и поскольку очередь отсортирована, они имеют наименьшие веса.

4. Создается новая вершина и ей приписывается вес, равный суммарному весу выбранных в п. 3 элементов. Сами элементы становятся вершинами бинарного дерева

– сыновьями созданной вершины – отца. Бинарные ориентированные деревья, корнями которых были указанные вершины, объединяются в одно новое бинарное дерево.

5. Вершина – отец добавляется в очередь на место, соответствующее ее весу, две вершины - ее сыновей, удаляются из очереди. При этом в очередь, по существу, помещается бинарное дерево, корнем которого является вершина-отец, а сама очередь представляет собой ориентированный лес.

6. Повторяем п. 3 – п.5 до тех пор, пока в очереди не останется только одна вершина – корень сформированного в процессе работы алгоритма полного бинарного дерева, листья которого помечены символами входного алфавита.

7. Одной дуге, исходящей из вершины-родителя (например, правой), ставится в соответствие метка 1, другой (левой) – метка 0. При компьютерной реализации алгоритма для хранения меток используется один бит.

Замечание. Шаг 7 ,то есть присваивание меток дугам графа можно выполнять и в процессе построения дерева.

8. Код каждого символа – это *метка пути* от корня дерева в соответствующий этому символу лист. Под *меткой пути* ведущего из вершины $\underline{v_i}$ в вершину $\underline{v_j}$, понимается слово в алфавите {0,1}, полученное соединением меток дуг пути в порядке их следования, пути нулевой длины не рассматриваются.

После обработки всего входного массива самые часто встречающиеся символы будут иметь самые короткие метки (для хранения этих меток требуется меньше всего места), а самые редко встречающиеся символы — более длинные метки.

Обычно сразу после построения дерева строится таблица Хаффмана, содержащая символы и соответствующие им коды. Содержимое этой таблицы обычно хранится в виде связного списка.

Рассмотрим пример построения бинарного кодового дерева Хаффмана.

Пусть дана строка «Veni, vidi, vici», в ASCII коде каждый символ занимает 8 бит, то есть на каждый печатный знак тратится по одному байту. Это означает, что вся строка целиком занимает 16*8 = 128 бит памяти, пробелы учитываются.

1. Вычислим частоту появления символов, заметим, что в случае больших текстов лучше вычислять вероятность появления символов. Запишем результат в виде таблицы

частота	*3*	*1*	*1*	*5*	*2*	*2*	*1*	*1*
символ	*‘v’*	*‘e’*	*‘n’*	*‘i’*	*‘,‘*	*‘_’* *	*‘d’*	*‘c’*

* символом ‘_’ обозначим пробел.

2. Сортируем полученный массив символов:

частота	*1*	*1*	*1*	*1*	*2*	*2*	*3*	*5*
символ	*‘e’*	*‘n’*	*‘d’*	*‘c’*	*‘ ‘*	*‘,’*	*‘v’*	*‘i’*

Формируем очередь, элементами очереди являются элементы отсортированного массива символов вместе с весами, например, запись *E/1* означает символ *E* с весом *1*.

Каждый элемент очереди считаем изолированной вершиной графа – корнем бинарного дерева ориентированного нулевой высоты, а все множество вершин считаем лесом (рис.1.).

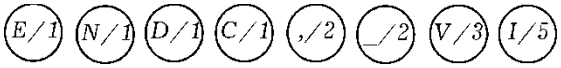

Рис.1.

3. Из головы очереди последовательно выбираем два элемента. *E/1* и *N/1*.

4. Создаем новую вершину $\underline{v_1}$ и приписываем ей вес, равный суммарному весу выбранных в п. 3 элементов ($\underline{v_1/2}$) (рис.2.).

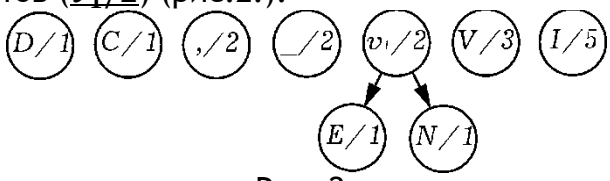

Рис. 2.

Повторяем шаги 3 и 4 до тех пор, пока в очереди не останется только одна вершина - корень сформированного бинарного ориентированное дерева (рис. 3 – 8).

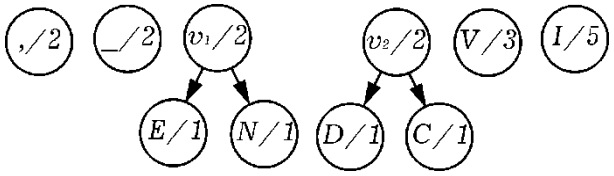

Рис. 3.

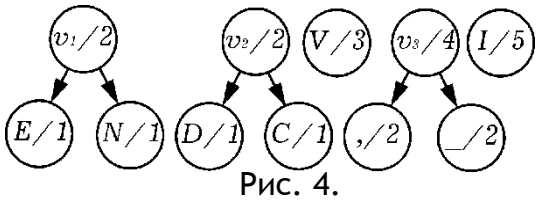

Рис. 4.

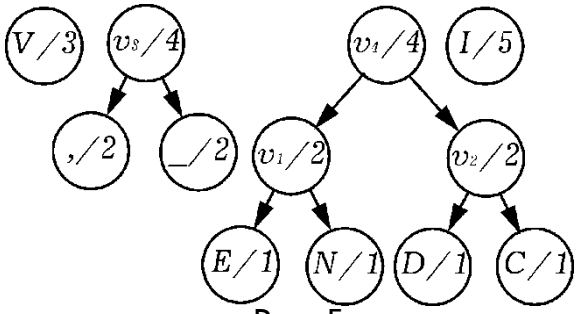

Рис. 5.

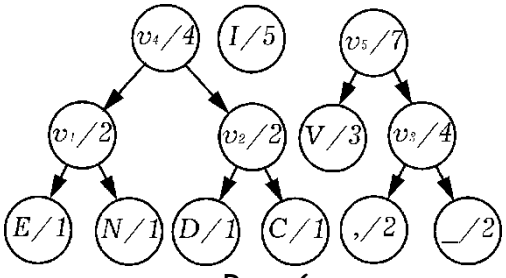

Рис. 6.

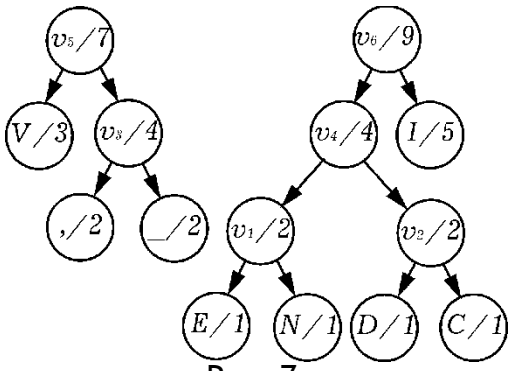

Рис. 7.

На рис. 8 показано построенное полное бинарное ориентированное дерево.

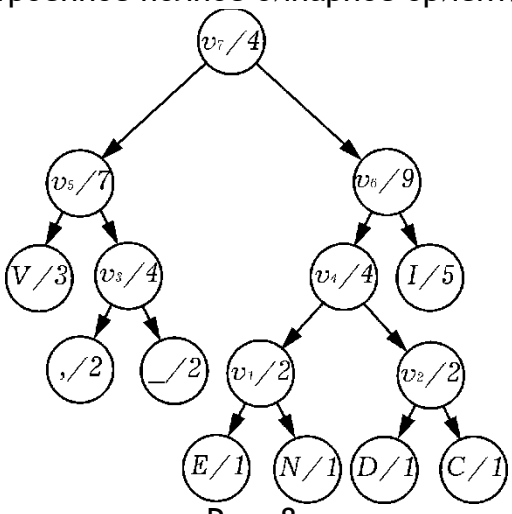

Рис. 8.

Далее присвоим метки дугам графа, правой дуге, исходящей из вершины - родителя поставим в соответствие метка 1, левой – метка 0. Получим кодовое бинарное ориентированное дерево (рис.9).

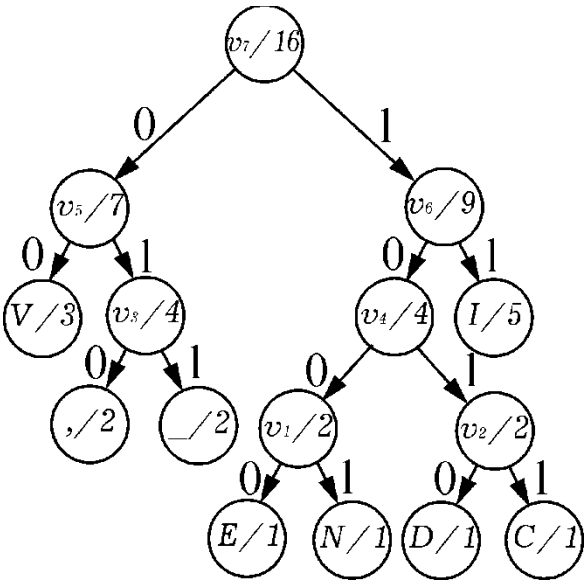

Рис. 9. Кодовое бинарное дерево

Код каждого символа есть метка пути от корня дерева вершины v_7 в соответствующий этому символу лист. Так, например, код символа ‘c’ – 1011, ему соответствует путь графе $v_7 \rightarrow v_6 \rightarrow v_4 \rightarrow v_2 \rightarrow c$.

Запишем таблицу кодовых символов.

‘v’	‘e’	‘n’	‘i’	‘ ,‘	‘ ’	‘d’	‘c’
00	1000	1001	11	010	011	1010	1011

Оценим размер закодированной фразы «Veni, vidi, vici», напомним, что 0 или 1 занимают один бит.

‘v’	**‘e’**	**‘n’**	**‘i’**	**‘ ,‘**	**‘ ’**	‘v’	‘i’	‘d’	‘i’	‘ ,‘	‘ ’	**‘v’**	**‘i’**	**‘c’**	**‘i’**
00	1000	1001	11	010	011	00	11	1010	11	010	011	00	11	1011	11

Вся строка целиком занимает 44 бита памяти, исходная строка занимала 128 бит памяти.

При построении кодового бинарного ориентированного дерева может наблюдаться некоторый «произвол», то есть, код одного и того же символа может быть разным при различных вариантах построения кодового дерева при одинаковой длине кода всего входного текста (массива символов). Это различие обуславливается различным порядком исходных символов с одинаковыми весами в отсортированном массиве символов. В рассмотренном выше примере символы ‘e’, ‘n’, ‘d’ и ‘c’ имеют одинаковую частоту, равную 1, а их порядок может быть произвольным в отсортированном по весам массиве. При показанном расположении символов (‘e’→‘n’→ ‘d’→ ‘c’) код символа ‘e’ – 1000, а код символа ‘c’ – 1011. Если поменять здесь местами, например, ‘e’ и ‘c’, то коды символов будут следующими: ‘e’ – 1011, ‘c’– 1000.

Из всех возможных вариантов кодировки лучшим будет код с наименьшей дисперсией, которая показывает, насколько сильно отклоняются длины индивидуальных кодов от их средней величины. Однако, дисперсия оказывает влияние на качество кодировки в случае, если этот сжатый файл будет потом передаваться по линиям связи. В этом случае наиболее более предпочтительны Коды Хаффмана с малой дисперсией только [1].

Недостатки алгоритма Хаффмана

Алгоритм Хаффмана малоэффективен для файлов маленьких размеров (если размер файла сопоставим с размером кодировочной таблицы) и бинарных файлов.

Классический алгоритм Хаффмана на основе кодового дерева требует хранения кодового дерева, это увеличивает его трудоемкость.

Для восстановления содержимого сообщения декодер должен знать таблицу частот, которой пользовался кодер.

Требуется два прохода для кодировки, первый проход – это составление таблицы частот и построение дерева, второй проход – это собственно кодирование.

Заключение

В статье приведен краткий обзор методов сжатия данных. Подробно разобран алгоритм Хаффмана сжатия информации на основе двоичных кодирующих деревьев. На примере кодирования фразы «Veni, vidi, vici» показана эффективность алгоритма Хаффмана. В рамках примера показана неоднозначность кодировки каждого символа в зависимости от расположения символов одинаковой частотой появления. Следует отметить, что подобная неоднозначность не влияет на качество сжатия при хранении данных. Также описаны недостатки данного метода. Заметим, что алгоритм Хаффмана достаточно часто используется для сжатия данных, как и в реляционных базах данных, так и в системах, работающими с большими данными.

ССЫЛКИ НА ИСТОЧНИКИ

1. Сэломон Д. Сжатие данных, изображения и звука. — М.: Техносфера, 2004. — С. 368.
2. Алгоритмы сжатия данных без потерь : учебное пособие для вузов / Е. Р. Пантелеев, А. Л. Алыкова. — Санкт: Лань, 2021. — 172 с. : ил.
3. Ватолин Д., Ратушняк А., Смирнов М., Юкин В.В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. - М.: ДИАЛОГ-МИФИ, 2003. - 384 с.
4. Габидулин, Э. М., Пилипчук Н. И. Лекции по теории информации : учебное пособие - М. : Изд-во МИФИ, 2007. - 213 с. : ил.. табл.
5. Белоусов А. И., Ткачёв С. Б. Дискретная математика : учебник для . (Сер. Математика в техническом университете; Вып. XIX) 6-е изд. - М. : Изд-во МГТУ им. Н. Э. Баумана, 2020. - 703 с. : ил.

***Marina S. Vinogradova*,**
Candidate of Physical and Mathematical Sciences, Associate Professor, Moscow State Technical University named after N.E. Bauman, Moscow
m-s-vinogradova@yandex.ru
***Olga S. Tkacheva*,**
Junior researcher, Institute of Control Sciences V.A. Trapeznikov Academy of Sciences, approved by the Presidium of the Russian Academy of Sciences, Moscow
tkolga17@gmail.com

Data compression. Huffman algorithm.

Abstract. The article considers data compression algorithms. A brief overview of lossy and lossless data compression methods is given. The Huffman data compression algorithm is described in detail, using terms from graph theory. An example of the operation of the algorithm for encoding the phrase "Veni, vidi, vici" based on a binary tree is considered. The efficiency and disadvantages of this data compression method are shown. The article is provided with illustrations.

Keywords: data compression, Huffman algorithm, binary trees, coding.