Уфа: УГАТУ Вестник УГАТУ 2001. № 2 (4). С. 97–104

УДК 681.3

МОДЕЛИ И АЛГОРИТМЫ СЖАТИЯ ИНФОРМАЦИИ

Ю. С. КАБАЛЬНОВ, И. В. ПАВЛОВ

Факультет информатики и робототехники УГАТУ
Тел: (3472) 23 78 76 E-mail: informatic@ugatu.ac.ru

Обсуждаются модели и алгоритмы сжатия информации. На основе предложенных моделей рассмотрена модификация алгоритма Лемпела–Зива, позволяющая существенно повысить степень сжатия исходных данных при высокой скорости их восстановления в процессе распаковки. Разработан алгоритм поиска совпадающих последовательностей символов для алгоритма Лемпела–Зива на основе математического аппарата бинарных деревьев

Модели представления информации; алгоритмы сжатия информации; кодирование данных; бинарные деревья

ВВЕДЕНИЕ

В настоящее время для снижения затрат на передачу и хранение информации все чаще используется ее сжатие. Возможность сжатия информации объясняется имеющейся информационной избыточностью в передаваемых сообщениях, обусловленной наличием повторяющихся или коррелированных частей в передаваемых сообщениях.

Алгоритмы сжатия информации делятся на два основных класса: сжатие с потерями и сжатие без потерь информации. Сжатие с потерями приводит к некоторой потере информации в процессе ее сжатия без заметного ухудшения качества воспроизведения и восприятия полученного информационного сообщения органами чувств человека. К такой информации можно отнести звуковые сообщения, графические данные и видеоизображения.

Сжатие без потерь не допускает потери информации. Примерами такой информации являются различные виды документальных и текстовых сообщений, компьютерные программы, информация, хранящаяся в базах данных, и т. п.

Часто в информационных системах один и тот же блок данных многократно передается различным получателям. В этом случае основными факторами, влияющими на выбор алгоритма сжатия, являются скорость процесса распаковки и степень сжатия. При этом скорость процесса упаковки особой роли не играет, так как сжатие обычно осуществляется только один раз. Так, например, в глобальной компьютерной сети Интернет данные можно один раз разместить (в сжатом виде) на одном из серверов Интернета, а затем эти данные могут быть считаны и распакованы конечными пользователями (получателями информации). Применяемые в этом случае алгоритмы сжатия данных должны иметь высокую степень сжатия и высокую скорость распаковки при невысоких требованиях к необходимым для распаковки ресурсам компьютера конечного пользователя.

Проанализируем основные алгоритмы сжатия данных, которые могут быть использованы в качестве основы для создания эффективного алгоритма сжатия данных, обладающего отмеченными выше свойствами.

Принято процесс сжатия делить на два этапа: моделирование, в результате которого определяются вероятности появления значений символов во входном потоке, и кодирование, в результате которого исходный поток символов преобразуется в сжатый (т. е. с меньшим числом символов) поток [1].

В настоящее время существует несколько основных алгоритмов моделирования для сжатия без потерь: алгоритм Лемпела — Зива (LZ); методы контекстуального предсказания (prediction by partial match, PPM); алгоритм "Burrows Wheeler Transform" (BWT).

Сущность алгоритма Лемпела – Зива (LZ) состоит в том, что символы (или группы символов) заменяются указателем на то место, где они в тексте уже ранее появлялись [1]. Достоинствами данного алгоритма являются высокая скорость распаковки и низкие требования к объему оперативной памяти компьютера пользователя. Недостатком алгоритма LZ является то, что он не обеспечивает высокой степени сжатия, такой, например, как у алгоритма PPM.

Методы контекстуального предсказания (PPM) базируются на предсказании появления символов во входном потоке. Алгоритм PPM для каждой позиции в тексте рассматривает контекст (группу из нескольких предыдущих символов) и определяет вероятности появления различных символов для данного контекста на основе статистического анализа появлений символов в уже просмотренном тексте в позициях с таким же контекстом [1]. Используя полученное распределение вероятностей появления символов, поступающий символ кодируется с помощью арифметического кодирования. Достоинством алгоритма PPM является сравнительно высокая степень сжатия, однако это сопровождается низкой скоростью распаковки и повышенными требованиями к объему оперативной памяти компьютера пользователя.

Алгоритм BWT – это алгоритм обратимой перестановки символов во входном потоке, позволяющий достаточно эффективно сжимать полученный в результате преобразования блок данных [2]. Алгоритм BWT обладает высокой скоростью упаковки и распаковки. К недостаткам алгоритма BWT можно отнести то, что для целого ряда типов данных, в частности для исполняемых файлов, степень сжатия хуже, чем у алгоритмов PPM и LZ.

Совместно с каждым из описанных выше алгоритмов моделирования, как уже было отмечено выше, используются алгоритмы кодирования. С помощью алгоритмов кодирования происходит формирование кодовых последовательностей потока сжатых данных с использованием полученных на этапе моделирования распределений вероятностей появления различных символов. В настоящее время используются два основных алгоритма кодирования: кодирование Хаффмана и арифметическое кодирование [3].

Принцип действия кодирования Хаффмана основывается на том, чтобы поставить в соответствие каждому символу определенную последовательность битов, длина которой определяется в соответствии с вероятностью данного символа [4]. При этом часто встречаемые символы получают более короткие кодовые последовательности. Другим наиболее применяемым методом кодирования является арифметическое кодирование, которое обладает меньшей избыточностью по сравнению с кодированием Хаффмана [3]. Следует отметить, что скорость процесса арифметического кодирования меньше скорости кодирования Хаффмана, так как арифметическое кодирование использует относительно медленные операции деления и умножения. Однако в настоящее время разработаны модификации алгоритмов арифметического кодирования, которые позволяют сократить количество наиболее медленных операций деления, что сильно повышает скорость работы данного алгоритма.

Из проведенного анализа видно, что ни один из перечисленных методов не обеспечивает одновременно высокую степень сжатия и высокую скорость распаковки, что затрудняет их использование в целом ряде практических областей, для которых характерна работа в режиме реального времени (on-line) с большими объемами запрашиваемой информации, в частности, открытое (или дистанционное) образование, системы информационной поддержки научных исследований, проектирование и управление сложными объектами. Поэтому возникает потребность в усовершенствовании существующих методов в направлении увеличения степени сжатия известных алгоритмов, обеспечивающих быструю распаковку, или путем повышения скорости распаковки известных алгоритмов, которые обеспечивают высокую степень сжатия.

Целью настоящей работы является разработка алгоритма сжатия на основе алгоритма Лемпела–Зива, который обеспечивает более высокую степень сжатия по сравнению с существующими модификациями алгоритма Лемпела–Зива. При этом ставится задача сохранить такие свойства алгоритма Лемпела–Зива, как высокая скорость распаковки и низкие требования к необходимому для распаковки объему оперативной памяти.

1. ПОСТАНОВКА ЗАДАЧИ ПОИСКА СОВПАДАЮЩИХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Алгоритм Лемпела–Зива использует тот факт, что последовательности символов часто повторяются в тексте. Эти повторяющиеся последовательности заменяются указателем на то место, где они в тексте уже ранее по-

являлись [1]. Одной из форм такого указателя может быть пара (n, m), которая ссылается на строку длиной m символов, начинающуюся с позиции n. Например, последовательность "абвабгбваб" может быть закодирована как "абв(1,2)г(2,4)".

Допустим, имеется исходная последовательность символов алфавита D:

$$A = a_0 \dots a_{N-1},$$

где N — размер последовательности. Пусть $B(n, m)$ — подпоследовательность A длиной m символов, начинающаяся с позиции n:

$$B(n, m) = a_n \dots a_{n+m-1}$$

и $E(n, m) = \{k_1, \dots, k_Z\}$ — множество всех чисел, для которых верно $B(n, m) = B(k_i, m)$ и $k_i < n$. Это множество содержит позиции всех строк длины m, предшествующих строке $B(n, m)$ и совпадающих с ней.

Если множество $E(n, m)$ содержит хотя бы один элемент, то мы можем закодировать строку $B(n, m)$ с помощью пары (e, m), где $e \in E(n, m)$. В большинстве реализаций алгоритма Лемпела–Зива позиция кодируется как разность между позициями n и e. Будем называть эту разность смещением. При этом из всех возможных вариантов для позиции e выбирается вариант с наименьшим смещением:

$$e = \max(E(n, m)).$$

Этот подход создает такое распределение вероятностей, при котором меньшие значения смещения обычно имеют более высокую вероятность появления, что обеспечивает более экономное кодирование по сравнению со случайным выбором позиции e из множества $E(n, m)$.

Процесс сжатия методом Лемпела–Зива можно описать как обратимую трансформацию исходной последовательности $A = a_0 \dots a_{N-1}$, содержащей символы некоторого алфавита D, в последовательность $F = f_0 \dots f_{Z-1}$ символов нового алфавита D'. Алфавит D' содержит все символы исходного алфавита D, а также новые составные символы, состоящие из пары чисел (смещение, длина). Далее будем называть символы нового алфавита просто символами, а символы исходного алфавита D, входящие в алфавит D', — одиночными символами.

Данную трансформацию можно называть разбиением, так как мы по сути дела разбиваем исходную последовательность A на цепочки символов и заменяем эти цепочки символами нового алфавита.

Заметим, что описанное разбиение можно производить различными способами. Поясним это на примере следующей последовательности символов:

Символы	абвг01	...	абвг12	...	абвг02
Позиция	0		1000		1010

Приведем варианты двух разбиений, которые можно использовать для кодирования данной последовательности:

{ а б в г 0 1 ... (1000, 4) 1 2 ... (1010, 5) 2 };
{ а б в г 0 1 ... (1000, 4) 1 2 ... (10, 4) 0 2 }.

Эти разбиения отличаются только в отношении к последовательности "абвг0", начинающейся с позиции (1010). Выбор разбиения, которое даст наибольшее сжатие, зависит от того, как мы кодируем полученные разбиения. Очевидно, что важную роль играет способ кодирования длин цепочек, смещений и одиночных символов.

Как было показано выше, вероятность появления короткого смещения обычно больше вероятности появления длинного смещения. А значит, короткие смещения кодируются с использованием меньшего числа битов, чем длинные смещения. Поэтому возможно, что кодирование последовательности "абвг0", начинающейся с позиции (1010), с помощью двух символов {(10, 4), 0} может обеспечить более высокую степень сжатия, чем кодирование той же последовательности с помощью одного символа (1010, 5). Такая ситуация возможна, так как количество сэкономленных битов при кодировании короткого смещения (10) вместо длинного смещения (1010) может превысить количество битов, необходимых для кодирования символа '0'.

Для сравнения различных вариантов разбиения можно использовать некоторую функцию $P(R)$, которая каждому разбиению R ставит в соответствие числовое значение, которое характеризует качество данного разбиения. Очевидно, что в качестве функции $P(R)$ целесообразно использовать функцию, которая определяет количество информации, необходимое для кодирования трансформированной последовательности. Тогда задача поиска наилучшего разбиения сводится к минимизации функции $P(R)$:

$$P(R) \to \min_{R \in G},$$

где G — множество всех допустимых разбиений для исходной последовательности A.

Для нахождения наилучшего разбиения необходимо решить задачу поиска всех совпадающих последовательностей, которые потенциально могут попасть в итоговое разбиение. Дадим формальное определение задачи поиска совпадающих последовательностей.

Пусть M — максимальная длина одной цепочки символов, т. е. максимальное значение для величины длины в символе (позиция, длина). Необходимо для каждого значения позиции n: $0 \leq n < N$ в исходной последовательности символов A и для каждой длины строки m: $1 \leq m \leq M$ найти ближайшее совпадение $T(n, m)$, если оно есть.

Под ближайшим совпадением $T(n, m)$ мы подразумеваем позицию ближайшей строки $B(e, m)$ из всех строк, совпадающих со строкой $B(n, m)$ и предшествующих ей.

Простейшим методом решения данной задачи поиска является последовательный поиск, когда для каждой позиции n: $0 \leq n < N$ осуществляется сравнение строки, начинающейся с позиции n, со строками, начинающимися с позиций n_1: $0 \leq n_1 < n$. Данный алгоритм требует приблизительно N^2 шагов для решения поставленной задачи.

В настоящее время для поиска совпадений наиболее часто используется алгоритм на основе хэш-цепочек [5]. Этот алгоритм использует некоторую хэш-функцию для вычисления хэш-кода (значение хэш-функции) по нескольким первым байтам текущей строки. Для каждого возможного хэш-кода создается связанный список. Этот список содержит все позиции в тексте, где хэш-функция дает данное значение. Для поиска строки, совпадающей с текущей строкой, вычисляется хэш-код, и в списке, соответствующем данному хэш-коду, осуществляется поиск совпадения максимальной длины.

Недостатком алгоритма на основе хэш-цепочек является возможность возникновения коллизий, когда разным строкам соответствует одно и то же значение хэш-функции. При этом эти строки попадают в один список, что вызывает замедление проведения поиска. Также замедление поиска может быть вызвано наличием большого количества строк, у которых совпадают все символы, которые участвуют в определении хэш-значения. Алгоритм, использующий хэш-цепочки, хорошо работает в случае, когда словарь для поиска мал, но для поиска в большом словаре он не обеспечивает достаточной производительности.

В связи с этим возникает необходимость в разработке алгоритма поиска, который отвечает поставленным требованиям и сложность которого линейно растет с увеличением размера входной последовательности. Этому требованию по сложности поиска удовлетворяют алгоритмы цифрового поиска [5]. Поэтому в статье предлагается использовать алгоритмы цифрового поиска в качестве основы для создания алгоритма поиска, отвечающего всем сформулированным выше требованиям.

2. АЛГОРИТМ ПОИСКА СОВПАДЕНИЙ НА ОСНОВЕ ДЕРЕВА ЦИФРОВОГО ПОИСКА

Пусть имеется исходная последовательность (строка) A символов алфавита D:

$$A = a_0 a_1 \dots a_{N-1}.$$

Будем рассматривать данную последовательность A как последовательность бинарных символов A', полученную из последовательности A заменой каждого символа a_n соответствующей последовательностью бинарных символов:

$$A' = b_0 b_1 \dots b_{N*S-1},$$

где S — количество бинарных символов, необходимых для кодирования исходных символов алфавита D.

Требуется для строки $K = a_f a_{f+1} \dots a_{f+M-1}$ найти длину максимального совпадения LEN и позиции всех оптимальных совпадений $POS[m]$, где $1 \leq m \leq LEN$.

Строку K также будем рассматривать в двоичном виде:

$$K' = b_{f*S} \dots b_{(f+M)*S-1} = k_0 \dots k_{M*S-1}.$$

В соответствии с принципами формирования алгоритмов цифрового поиска [5] введем в рассмотрение бинарное дерево. Структура дерева задается с помощью множества узлов, где каждому узлу соответствует поддерево, для которого этот узел является корневым и при этом каждый узел содержит следующие поля: SKIP — количество пропущенных битов; LAST — указатель на последнюю строку поддерева; CHILD[2] — указатели на два дочерних узла.

Поле SKIP содержит число начальных битов, которые совпадают у всех строк, представленных в поддереве данного узла. Рассматриваются только биты, которые начинаются с позиции, соответствующей данному узлу.

Поле LAST содержит указатель на позицию строки в исходном массиве, которая была включена в дерево последней из всех строк

данного поддерева. Это поле используется для проверки на совпадение пропущенных битов с соответствующими битами текущей строки. Эта проверка проводится между текущей строкой и строкой, на которую указывает данное поле. Поле LAST также используется для заполнения массива оптимальных совпадений.

Поля CHILD[0] и CHILD[1] являются составными и содержат следующие поля: TAG — определяет значение поля LINK; LINK — указатель на дочерний узел.

Если значение поля TAG равно 0, то LINK содержит указатель на строку в тексте, в противном случае LINK содержит указатель на дочерний узел.

В качестве примера рассмотрим строку "ABCAD". Все уникальные подстроки минимальной длины для последовательности "ABCAD" показаны в табл. 1, а бинарное дерево поиска, содержащее подстроки последовательности "ABCAD", показано на рис. 1.

Таблица 1

Позиция	Символы	Двоичное представление
0	AB	0100 0001 0100 00
3	AD	0100 0001 0100 01
1	B	0100 0010
2	C	0100 0011
4	D	0100 01

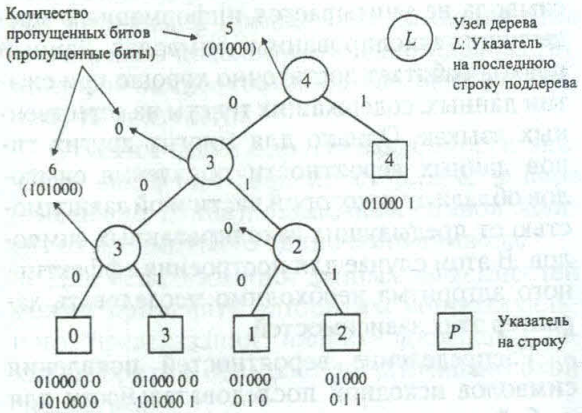

Рис. 1. Бинарное дерево поиска для строки "ABCAD"

Для указания корневого элемента будем использовать переменную HEAD типа CHILD, содержащую поля TAG и LINK. Также будем использовать переменную EMPTY, которая указывает, содержит ли наш объект хотя бы одну строку. До появления первой строки присваиваем EMPTY $\leftarrow 1$, что означает отсутствие строк в объекте.

Опишем предлагаемый алгоритм поиска и вставки строки (рис. 2).

В качестве входных параметров выступают: A — исходная последовательность символов; f — позиция вставляемой строки в последовательности A (f последовательно принимает значения от 0 до $N-1$). В качестве выходных параметров выступают: LEN — длина совпадения с максимальной длиной; $POS[0 \ldots LEN]$ — массив позиций оптимальных совпадений, где переменная $POS[m]$ должна содержать позицию оптимального совпадения для строки длиной m.

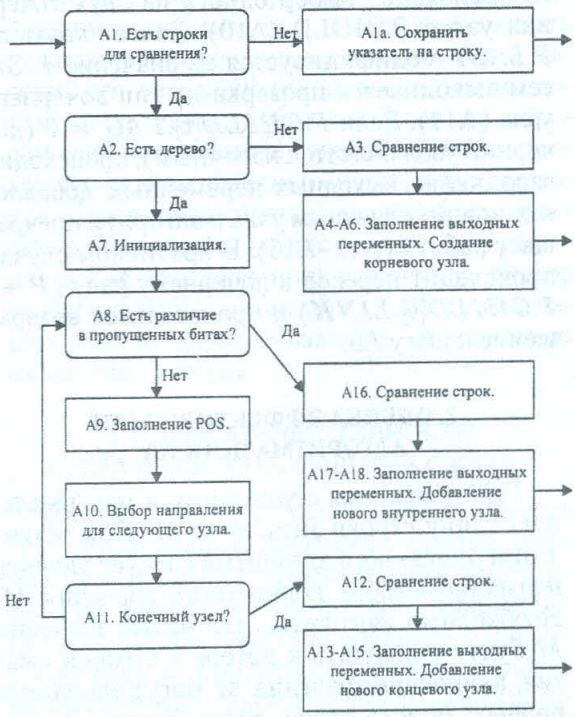

Рис. 2. Алгоритм поиска совпадающих последовательностей

Как следует из рис. 2, работа алгоритма начинается с проверки на наличие строк для сравнения (A1). Если еще нет ни одной строки, то происходит сохранение указателя на текущую строку и алгоритм прекращает работу. В противном случае проверяется наличие дерева, содержащего строки для сравнения. Если дерево не создано (есть всего одна строка для сравнения), происходит сравнение текущей строки с имеющейся строкой (A3), заполнение выходных переменных, создание корневого узла дерева, содержащего данные две строки, и алгоритм прекращает работу (A4–A6). В противном случае (если дерево создано) производится инициализация необходимых для перемещения по дереву переменных (A7), в частности, переменная P получает указатель на корневой узел дерева, и начинается выполнение основного цикла.

Работа с текущим узлом начинается со сравнения $P.SKIP$ битов у текущей строки и строки $P.LAST$ (A8). Если в этих битах есть различия, то происходит заполнение выходных переменных, добавление нового внутреннего узла в текущую ветку и алгоритм прекращает работу (A16 – A18). В противном случае происходит заполнение переменных POS значением $P.LAST$ для всех длин, соответствующих переходам через символьную границу в пропущенных битах (A9), затем определяется значение бита b, на основе которого происходит выбор одного из двух дочерних узлов P.CHILD (A10). Далее указатель $P.LAST$ модифицируется на значение f. Затем выполняется проверка на тип дочернего узла (A11). Если $P.CHILD[b].TAG = 0$ (дочерний узел является конечным), происходит заполнение выходных переменных, добавление нового конечным узла и алгоритм прекращает работу (A12–A15). В противном случае происходит переход к дочернему узлу ($P \leftarrow P.CHILD[b].LINK$) и производится возвращение к шагу A8.

3. ОЦЕНКА ЭФФЕКТИВНОСТИ АЛГОРИТМА ПОИСКА

При поиске мы ограничивали максимальную длину строки значением M. При реализации описанного алгоритма следует уделить внимание выбору конкретного значения M. Необходимо учитывать, что малые значения M будут приводить к потере в степени сжатия, а большие значения M могут замедлить процесс поиска строк, когда данные содержат длинные повторяющиеся блоки. Для достижения высокого уровня сжатия достаточно использовать такое значение M, которое превышает большинство длин повторяющихся последовательностей. Например, для текстов на естественных языках значение $M = 20$ обычно является достаточным для получения высокой степени сжатия.

Оценим объем памяти, необходимый для работы алгоритма.

Количество узлов в дереве равно количеству уникальных подстрок, т. е. размеру окна K. Размер окна — это размер последовательности, в которой осуществляется поиск совпадающей строки.

Количество памяти в битах для хранения одного узла бинарного дерева равно

$$W = 3\log_2 K + \log_2 M' + 2,$$

где M' — максимальная длина строки в битах.

Представленный алгоритм обеспечивает нахождение оптимальных совпадений, ограниченных длиной M' битов, используя не более чем $(M't)$ операций, где t — количество операций, необходимых для обработки одного узла дерева. Но обычно на поиск затрачивается порядка $t\log_2 K$ операций. В итоге получаем примерное количество инструкций для поиска совпадений для всего текста:

$$C = Nt\log_2 K,$$

где N — размер текста; t — количество инструкций на обработку одного узла; K — размер словаря. Видно, что количество инструкций C для поиска совпадений находится в линейной зависимости от размера текста N.

4. КОДИРОВАНИЕ ПРЕОБРАЗОВАННОЙ ПОСЛЕДОВАТЕЛЬНОСТИ

Второй этап процесса сжатия информации — это кодирование последовательности символов, полученной из исходной последовательности заменой повторяющихся цепочек символов указателями. На этом этапе существующие программы сжатия данных обычно используют алгоритмы, которые спроектированы для эффективного кодирования сообщений источника без памяти. Таким образом, при кодировании очередного символа не учитывается информация о предыдущих закодированных символах. Данный подход работает достаточно хорошо при сжатии данных, содержащих тексты на естественных языках. Однако для многих других типов данных вероятности появления символов обладают некоторой частичной зависимостью от предыдущих закодированных символов. В этом случае для построения эффективного алгоритма необходимо исследовать характер этих зависимостей.

Распределение вероятностей появления символов исходной последовательности для любой позиции зависит от контекста из нескольких предыдущих символов. Рассмотрение больших по длине контекстов является причиной относительно низкой скорости работы методов контекстуального предсказания (PPM). Но после преобразования исходной последовательности алгоритмом Лемпела–Зива большая часть избыточности информации устранена на первом этапе. Поэтому сохранившиеся зависимости носят локальный характер, что создает возможность использования методов контекстуального предсказания для преобразованной по-

следовательности без существенной потери в скорости работы.

В результате исследования авторами различных типов данных с помощью разработанной программы сжатия данных были обнаружены несколько основных зависимостей. Перечислим их и покажем, как можно эффективно использовать эти зависимости при кодировании.

Для некоторых типов данных значения смещения в паре (смещение, длина) у символов, расположенных близко друг к другу, имеют тенденцию повторяться. Для того чтобы использовать этот факт, можно поддерживать таблицу из нескольких наиболее недавно используемых значений смещений и ввести в алфавит дополнительные символы, означающие выбор значения смещения из этой таблицы.

Символы нового алфавита можно разделить на классы: одиночный символ; пара (смещение, длина); пара (смещение по таблице, длина).

Кодировать такие составные символы можно поэтапно. Сначала кодируется тип (класс), а затем значение из данного класса. Причем тип символа часто обладает зависимостью от типов двух-трех предыдущих символов. Так как алфавит типов достаточно мал, можно реализовать быстрый алгоритм кодирования, учитывающий контекст из типов нескольких предыдущих символов. Для этого можно использовать цепь Маркова [6], в которой каждое состояние соответствует одному из контекстов.

Отметим также и другие зависимости: значение смещения зависит от длины в паре (смещение, длина); одиночный символ зависит от предыдущего одиночного символа;

Для использования данных зависимостей можно применять алгоритмы контекстуального предсказания низких порядков. Такие алгоритмы обладают достаточно высокой скоростью.

При создании эффективного алгоритма сжатия следует также учитывать степень статистической стабильности данных. Например, тексты на естественных языках достаточно стабильны, т. е. статистические показатели в различных участках практически совпадают. Но многие другие типы файлов не обладают такой стабильностью. Поэтому для того чтобы повысить степень сжатия, алгоритм должен быстро приспосабливаться к меняющимся данным.

Рассмотрим модель кодирования. Пусть $A = \{a_1 \dots a_N\}$ — некоторый алфавит. На вход модели подается последовательность символов

$$H = \{a_{i_1} \dots a_{i_K}\}.$$

Модель должна как можно точнее оценивать распределение вероятностей для символов алфавита A (до поступления очередного символа a_{i_j}). Для получения этих вероятностей обычно используется статистика появления предыдущих символов. Для того чтобы модель быстро приспосабливалась к меняющимся данным, она должна присваивать недавно закодированным символам более высокий вес, чем давно закодированным символам. Этого можно добиться, например, сбросом всей накопленной статистики после прохождения определенного количества символов. Более эффективным является метод деления пополам весов всех прошедших символов после определенного количества. Выбор коэффициентов, задающих скорость адаптации модели, должен зависеть от характера сжимаемых данных.

5. ОЦЕНКА ЭФФЕКТИВНОСТИ ПРЕДЛАГАЕМОГО АЛГОРИТМА СЖАТИЯ

Разработанный алгоритм сжатия данных был реализован в программе сжатия файлов. Для проверки эффективности программы был выбран набор файлов "Canterbury Corpus", который часто используется при тестировании алгоритмов сжатия данных. Этот набор содержит: текстовые файлы, файл электронной таблицы, графический файл и др. Общий размер файлов равен 2810784 байтам.

Для проверки, наряду с разработанной программой LZA, реализующей предлагаемый алгоритм сжатия, были использованы следующие известные программы: BZIP — программа, использующая алгоритм BWT; RAR — программа, использующая модифицированный алгоритм Лемпела–Зива с большим словарем и кодирование Хаффмана.

Проверка проводилась на компьютере с процессором K6-2+ 450 MHz. В процессе проверки производилось сжатие приведенного выше набора файлов с помощью предложенного метода и известных методов. Результаты сжатия сравнивались по четырем показателям: размер сжатых данных (степень сжатия), время упаковки, время распаковки и суммарное время, которое необходимо затратить для получения файлов в сжатом виде со скоростью 5 Кб/с и распаковку. Результаты данного эксперимента представлены в табл. 2.

Таблица 2

Программа	Размер файла, б	Время упаковки, с	Время распаковки, с	Время получения и распаковки, с
LZA	503126	15.16	0.70	101.32
BZIP	542710	6.04	1.32	109.86
RAR	593668	14.61	0.33	119.06

Полученные результаты показывают, что разработанный алгоритм на данном наборе обеспечивает уменьшение суммарного времени, необходимого для получения информации и ее распаковки, на 15% по сравнению с существующей модификацией алгоритма LZ.

ВЫВОДЫ

В статье предложены модели и алгоритмы сжатия информации при ее передаче пользователям в компьютерных сетях, базирующиеся на использовании алгоритма Лемпела–Зива сжатия информации. Повышение эффективности сжатия информации при использовании предложенных алгоритмов достигается за счет использования, в дополнение к алгоритму Лемпела–Зива, алгоритма цифрового поиска совпадающих последовательностей символов и методов контекстуального предсказания вероятностей появления символов.

На основе предложенных алгоритмов сжатия информации создан новый архиватор данных 7-zip. Программное обеспечение данного архиватора имеет открытую архитектуру, что позволяет добавлять поддержку новых методов архивирования и сжатия с помощью добавления соответствующих модулей. Данный архиватор доступен для пользователей по адресу www.7-zip.org.

Экспериментальная проверка эффективности разработанных алгоритмов показала, что они обеспечивают уменьшение суммарного времени, необходимого на получение информации и ее распаковку, в среднем на 10–20% по сравнению с существующими алгоритмами.

Разработанные алгоритмы целесообразно применять в информационных системах, для которых характерна работа в режиме реального времени (on-line) с большими объемами запрашиваемой информации, когда один и тот же блок данных многократно передается различным получателям. Примерами таких информационных систем являются информационно-обучающие системы открытого или дистанционного образования, системы информационной поддержки научных исследований, проектирования и управления сложными объектами.

СПИСОК ЛИТЕРАТУРЫ

1. **Bell T., Witten I. H., Cleary J. G.** Modelling for text compression // ACM Computing Surveys. 1989. V. 21, No 4. P. 557–591.
2. **Burrows M., Wheeler D. J.** A block-sorting lossless data compression algorithm // SRC Research Report 124. Palo Alto: Digital Systems Research Center, May 1994.
3. **Потапов В. Н.** Теория информации. Кодирование дискретных вероятностных источников. Новосибирск, 1999. 71 с.
4. **Юсупова Н. И., Миронов В. В.** Основы теории информации для информатиков. Уфа: УГАТУ, 2001. 160 с.
5. **Кнут Д. Е.** Искусство программирования. Т. 3. Сортировка и поиск. 2-е изд. М.: Вилиямс, 2000. 822 с.
6. **Вентцель Е. С., Овчаров Л. А.** Теория случайных процессов и ее инженерные приложения. М.: Наука, 1991. 383 с.

ОБ АВТОРАХ

Кабальнов Юрий Степанович, профессор, зав. кафедрой информатики УГАТУ. Дипл. инженер электронной техники (УАИ, 1971), д-р техн. наук по управлению в технических системах (УГАТУ, 1993). Исследования в области адаптивного и интеллектуального управления сложными техническими объектами.

Павлов Игорь Викторович, аспирант той же кафедры. Дипл. инж. по программному обеспечению вычислительной техники и автоматизированных систем (УГАТУ, 1998). Готовит диссертацию по эффективному сжатию данных.

На кафедре информатики УГАТУ в рамках ФЦП «Интеграция» осуществляется выпуск серии учебных пособий, охватывающих основные разделы информатики. В пособиях с системных позиций излагаются современные представления об информатике и перспективах ее развития. Пособия предназначены для изучения студентами технических вузов информатики в рамках начальной базовой подготовки и могут быть использованы также при первоначальном, самостоятельном изучении дисциплин информационного профиля

Кабальнов Ю. С., Ахметсафина Р. З., Карасев С. В. Компьютерные системы хранения информации. Уфа: УГАТУ, 2000. 151 с. ISBN 5-86911-297-4.

Рассматриваются теоретические основы, принципы построения компьютерных систем хранения информации, получивших в последнее время самое широкое распространение практически во всех областях человеческой деятельности. Большое внимание уделяется системному изложению различных по своему назначению и функциональным возможностям компьютерных систем хранения информации. Приводится большое число примеров, иллюстрирующих излагаемый материал.

Предназначено для студентов технических вузов, изучающих дисциплины «Базы данных», «Проектирование баз и банков данных» в рамках базового курса информатики, пособие может быть полезно также аспирантам и научным работникам при первоначальном (в том числе самостоятельном) изучении компьютерных систем хранения информации.

Кабальнов Ю. С., Лебедев В. А., Осипова Г. В. Языки и технологии программирования. Уфа: УГАТУ, 2000. 159 с. ISBN 5-86911-329.6.

Рассматриваются теоретические основы построения языков программирования, приводится их классификация и история развития языков, стилей и технологий программирования. Излагаются технологии разработки программных средств, приводятся критерии качества компьютерных программ. Рассматриваются методы, инструменты разработки программных средств, а также инструментальные среды их разработки. Анализируются основные стили программирования: процедурно- и функционально-ориентированный, модульный, логический, объектно-ориентированный и компонентный, играющие ключевые роли в современных технологиях программирования. Рассматриваются вопросы, связанные с языками и технологиями Internet, даются основные сведения о супер-ЭВМ и параллельном программировании, современных направлениях их развития.

Предназначено для студентов технических вузов, изучающих дисциплины «Информатика», «Компьютерные языки и программирование» и другие дисциплины информационного профиля, может быть полезно также аспирантам и научным работникам, желающим получить систематизированное представление о существующих языках и технологиях программирования.

Кабальнов Ю. С., Ахметсафин Р. Д., Строкина Ю. Г. Компьютерные сети. Уфа: УГАТУ, 2000. 105 с. ISBN 5-86911-320-2.

Дается понятие компьютерной сети, описываются общие компоненты всех типов компьютерных сетей, виды топологии сетей и задачи по управлению их работой. Приводятся краткие теоретические основы и принципы построения компьютерных сетей, традиционные и перспективные информационные технологии, используемые в локальных и глобальных компьютерных сетях. Рассматриваются вопросы кодирования и передачи данных по каналам связи, физические среды передачи (кабельные линии связи, оптоволоконные линии связи, беспроводная связь), вопросы управления потоком данных по каналам связи. Описываются модели и программное обеспечение функционирования компьютерных сетей, даются понятия протоколов, интерфейсов и сервисов сетей, рассматриваются две основных стандартных модели функционирования компьютерных сетей: OSI и TCP/IP. Рассматриваются варианты аппаратного обеспечения компьютерных сетей с использованием телефонных линий связи, цифровых сетей с интегрированным сервисом (ISDN), спутниковых систем связи. Приводятся примеры организации функционирования больших сетей (Internet), описывается одна из наиболее интересных услуг Internet, а именно IP-телефония.

Предназначено для студентов технических вузов, изучающих дисциплины «Компьютерные системы и сети», «Компьютерные сети» в рамках базового курса информатики.

В 2001–2002 гг. планируется издать:

Кабальнов Ю. С., Ахметсафина Р. З., Каримов Р. Р., Шехтман Л. И. Компьютерные системы отображения информации.

Кабальнов Ю. С., Сахабетдинов М. А., Тархов С. В., Кузьмина Е. А., Карчевская М. П. Введение в информатику.