

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Алгоритмы кодирования

Студент гр. 9303

Ахримов А.М.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Реализовать кодирование алгоритмом Хаффмана.

Задание.

3 вариант. Кодирование: статическое Хаффмана

Основные теоретические положения.

Один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности появления символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (H-дерево).

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.

6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Выполнение работы.

Данные можно передать через файл или терминал (для чтения из файла программе нужно передать в качестве аргумента названия файла).

Для считанных символов создаются объекты класса Node.

Класс Node хранит символ и его частоту встречаемости в тексте. Также для создания бинарного дерева и связывания между собой узлов класс хранит в себе указатели на два других объекта класса Node. Класс Node может хранить несколько символов (в одной строке типа `std::string`) и их общую частоту встречаемости.

Из указателей на объекты класса Node создаётся приоритетная очередь (`priority_queue` из `stl`). Приоритет определяется частотой встречаемости символа — чем реже символ встречается в исходном тексте, тем он приоритетнее.

Далее функция `create_tree()` создаёт бинарное дерево (объект класса `BinTree`). Она последовательно берёт два значения из приоритетной очереди и создаёт новый объект класса Node, складывая общую частоту двух символов, и вставляет новый объект в очередь. Функция завершает свою работу, когда в очереди останется один элемент. В конце функция создаст объект класса `BinTree` и даст ему указатель на оставшийся элемент в очереди.

Класс `BinTree` хранит указатель на объект класса Node, являющийся корнем бинарного дерева. У `BinTree` есть метод `coding`, который возвращает словарь закодированных символов. Метод рекурсивно проходит по бинарному дереву и присваивает листьям дерева двоичный код.

Таким образом, по словарю кодируется исходное сообщение.

Самая дорогая операция в алгоритме Хаффмана — это вставка нового элемента и сортировка при создании узла бинарного дерева. Так как в данной реализации используется приоритетная очередь, то сложность алгоритма Хаффмана равна $O(N \log N)$.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	aaggeeeqqqqwwwwwrrrttaa abbbqqqqwedsg	Binary Tree: btdsqeawrg = 38 btdsq = 15 btds = 7 b = 3 tds = 4 t = 2 ds = 2 d = 1 s = 1 q = 8 eawrg = 23 ea = 11 e = 5 a = 6 wrg = 12 w = 6 rg = 6 r = 3

		$g = 3$ Dictionary: $a = 101$ $b = 000$ $d = 00110$ $e = 100$ $g = 1111$ $q = 01$ $r = 1110$ $s = 00111$ $t = 0010$ $w = 110$ Encoded message: 101101111111111100100100100010101011 10110110110110111011101110001000101 0110110110100000000001010101110100 00110001111111
2.	3ahe7933vx!-dsfff eea	Binary Tree: 3h!xfa7edv -9s = 21 3h!xf = 9 3h = 4 3 = 3 h = 1 !xf = 5 !x = 2 ! = 1 x = 1 f = 3 a7edv -9s = 12

		$a7e = 6$ $a7 = 3$ $a = 2$ $7 = 1$ $e = 3$ $dv - 9s = 6$ $dv = 3$ $d = 1$ $v = 2$ $v = 1$ $= 1$ $-9s = 3$ $- = 1$ $9s = 2$ $9 = 1$ $s = 1$ Dictionary: $= 11011$ $! = 0100$ $- = 1110$ $3 = 000$ $7 = 1001$ $9 = 11110$ $a = 1000$ $d = 1100$ $e = 101$ $f = 011$ $h = 001$ $s = 11111$ $v = 11010$ $x = 0101$ Encoded message: 0001000001101100111110000000110100101010
--	--	--

		01110110011111011011011110111011011000
--	--	--

Выводы.

В ходе выполнения работы был изучен алгоритм Хаффмана. Были изучены его достоинства и недостатки. Был реализован алгоритм Хаффмана с применением приоритетной очереди и была установлена сложность такой реализации.