Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и информатики

Лабораторная работа №1 по дисциплине

Дискретная математика

Тема: «Кодирование»

Вариант 2 – Алгоритм Хаффмана

Выполнил студент гр.5030102/20202 Дрекалов Н. С.

Преподаватель Новиков Ф. А.

Санкт-Петербург

2024

Оглавление

[Формулировка задания 3](#_Toc178643758)

[Используемый язык программирования и системы сборки 3](#_Toc178643759)

[Описание алгоритма кодирования Хаффмана 3](#_Toc178643760)

[Подсчёт частот встречающихся символов и сохранение их в очереди приоритетов 3](#_Toc178643761)

[Построение дерева кодов Хаффмана и словаря из него 3](#_Toc178643762)

[Кодирование текста с использованием словаря кодов 4](#_Toc178643763)

[Описание алгоритма декодирования 4](#_Toc178643764)

[Считывание словаря кодов и битов из закодированного текста 4](#_Toc178643765)

[Демонстрация работы алгоритма 5](#_Toc178643766)

[Шаг 1. Подсчёт частот и сохранение в очереди приоритетов 5](#_Toc178643767)

[Шаг 2. Построение дерева кодов Хаффмана 5](#_Toc178643768)

[Шаг 4. Кодирование текста с использованием словаря кодов 7](#_Toc178643769)

[Область применения реализованного алгоритма 8](#_Toc178643770)

[Формат входных и выходных данных 9](#_Toc178643771)

[Сравнение результатов кодирования с равномерным кодированием (ASCII) 9](#_Toc178643772)

# Формулировка задания

В лабораторной работе требуется реализовать кодирование файла с помощью алгоритма Хаффмана

# Используемый язык программирования и системы сборки

* C++ 20
* CMake 3.28.3
* Ninja 1.11.1

# Описание алгоритма кодирования Хаффмана

Алгоритм получает на вход текст и состоит и следующих действий:

1. Подсчёт частот встречающихся символов и сохранение их в очереди приоритетов.
2. Построение дерева кодов Хаффмана и словаря из него
3. Кодирование текста с использованием словаря кодов и запись в файл

## Подсчёт частот встречающихся символов и сохранение их в очереди приоритетов

1. Создаем словарик символ – частота встречи.
2. Проходим по тексту и увеличиваем частоту встречи для каждого попавшегося символа
3. Проходим по словарику и сохраняем каждую пару как бинарное дерево с единственным узлом в очереди приоритетов, сортируя по частоте от меньшему к большему.

## Построение дерева кодов Хаффмана и словаря из него

1. Если очередь приоритетов имеет больше 1 элемента – выполняем пункт 2, иначе 3.
2. Достаем из очереди приоритетов 2 элемента left, right и создаем в очереди новое дерево, левое поддерево которого будет left, правое – right, а частота – суммарная частота left + right. Переходим к 1.
3. Обходим дерево в глубину, начиная с пустым кодом. Для каждого узла проверяем – если это лист, то мы дошли до символа и его нужно записать в словарик парой символ-код, иначе идем в левое поддерево, добавляя к коду справа “0”, и в правое, добавляя “1”.
4. В итоге получаем словарь, сопоставляющий каждому символу его код.

## Кодирование текста с использованием словаря кодов

1. Сохраним для начала дерево кодов – чтобы можно было потом декодировать текст. Для этого записываем в 1 байт число n – число закодированных символов. Далее записываем n раз данные о символе в виде символ (1 байт) + длина кода k (6 бит) + k бит самого кода.
2. Затем обходим каждый символ искомого текста и дописываем в закодированный текст код, который соответствует значению в словаре для этого символа. В конце добавляем дополнительный байт, в котором хранится число заполненных бит в предпоследнем байте.
3. Полученную закодированную строку записываем в файл.

# Описание алгоритма декодирования

Алгоритм получает на вход закодированный текст и состоит из следующих действий:

1. Считывание словаря кодов из закодированного текста
2. Раскодирование закодированного текста с использованием словаря кодов

## Считывание словаря кодов и битов из закодированного текста

1. Считываем словарь – считываем число закодированных символов n, затем n раз считываем данные – символ символ (1 байт) + длина кода k (6 бит) + k бит самого кода. Полученные данные записываем в словарик код-символ.
2. Считываем далее закодированные биты. Как только код – последовательность бит – будет совпадать с кодом какого-нибудь символа – записываем этот символ в результирующую строку.

# Демонстрация работы алгоритма

Закодируем строку “aabbbccd”, используя алгоритм кодирования Хаффмана

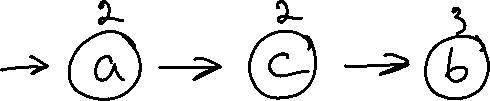
## Шаг 1. Подсчёт частот и сохранение в очереди приоритетов

a – 2

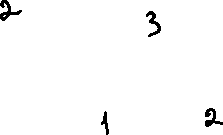
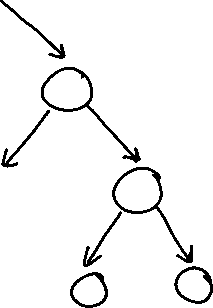
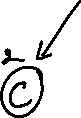
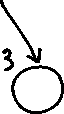
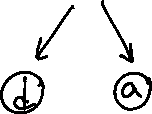
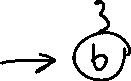
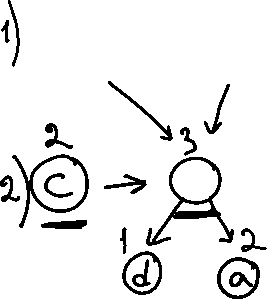
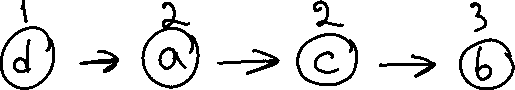
b – 3

c – 2

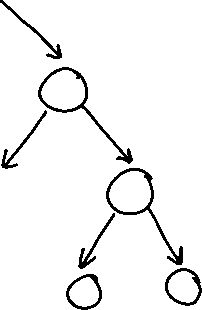
d – 1



## Шаг 2. Построение дерева кодов Хаффмана



##### Шаг 3. Построение словаря из дерева кодов Хаффмана



b – 0

c – 10



d – 110

a - 111

## Шаг 4. Кодирование текста с использованием словаря кодов

Кодирование дерева:

1 байт – число символов – 4: 00000100

Далее кодируем словарик – символ (8 бит) + длина кода (6 бит) + сам код

b – 01100010 + 000001 + 0 = 011000100000010

c – 01100011 + 000010 + 10 = 0110001100001010

d – 01100100 + 000011 + 110 = 01100100000011110

a – 01100001 + 000011 + 111 = 01100001000011111

Кодируем саму строку

aabbbccd = 111 111 0 0 0 10 10 110 = 1111110001010110

Итого получаем

00000100011000100000010011000110000101001100100000011110011000010000111111111110001010110

В последний байт дописываем нули и добавляем ещё один байт, хранящий число значимых битов в последнем байте – 3 (00000010)

Результат кодирования:

000001000110001000000100110001100001010011001000000111100110000100001111111111100010101100000000000010

Искомая строка – 8 байт, закодированная строка – 16 бит (2 байта). Закодированная с кодами строка – 13 байт.

Получили сжатие в 4 раза полезных данных и увеличение объема в ~1.5 раза всего файла.

# Область применения реализованного алгоритма

Алгоритм ставит в соответствие символу тем меньший код, чем чаще он встречается в искомом тексте. Таким образом, алгоритм работает **корректно** на текстах с неравномерным распределением частоты встречи каждого символа и **некорректно** на текстах, в которых частота встречи каждого символа примерно одинакова.

Отсюда можно сделать вывод, когда алгоритм полезен, а когда – нет.

* *«+»:* при кодировании текстовых файлов, содержащих осмысленный человеческий язык. Любой язык имеет неравномерное распределение по частоте встречаемых букв, а также чисто текстовые символы – это около 30% всех 8-битных кодов, что создает идеальные условия для использования алгоритма Хаффмана.
* *«*–*»:* при кодировании чисто бинарных файлов, например изображений. В таком случае в файле содержатся (с точки зрения 8-битных блоков char) случайные данные, и частота встречи каждого символа будет примерно одинаковая, что не даст возможность получить выгоду от сжатия (так как уменьшение длины одного кода идёт за счёт увеличения длины у другого).

# Формат входных и выходных данных

*Входные данные* – текст в файле с неравномерным распределением частот символов.

*Выходные данные* – бинарная строка в файле, представляющая собой закодированный с помощью [алгоритма Хаффмана](#_Описание_алгоритма_кодирования) изначальный текст.

# Сравнение результатов кодирования с равномерным кодированием (ASCII)

Для чистоты эксперимента будем учитывать записанный словарик.

Сравнивать результаты будем на книге «1984» Дж. Оруэлла. Будем считывать n первых символов и смотреть на степень сжатия данных. n будем менять от 10 до 10000 с шагом 100.

Результаты исследования показаны на графике.

Заметно, что степень сжатия сильно возрастает с 600 символов, что связано с тем, что примерно к этому моменту в тексте перестают появляться новые символы и становятся заметны разности частот между символами. После возрастания степень лишь немного увеличивается, иногда проседая (появление новых символов), что связано с тем, что с увеличением символов особо не меняется их относительная частота появления.

Также стоит отметить, что с помощью алгоритма Хаффмана удалось сжать исходный текст почти в 2 раза (на 43%).