Санкт-Петербургский Политехнический Университет
Высшая школа прикладной математики и вычислительной физики, ФизМех
01.03.02 Прикладная математика и информатика

Лабораторная Работа №1
тема "Кодирование информации"
вариант "Алгоритм Фано"
дисциплина "Дискретная математика"

Выполнил студент гр. 5030102/20201 Буслама Анис

7 октября 2024 г.

Содержание

1 Поставленная задача

1.1 Постановка задачи:

Реализовать систему, которая будет применять алгоритм Фано, к переданному ей текстовому файлу. Также необходимо поддержать обратную операцию. Программа должна поддерживать перечисленные операции по отдельности.
Для простоты предлагается совершать манипуляции с текстом, в котором присутствуют только ASCII символы.
Поддержать возможность вывода кодов на экран при кодировании и декодировании информации для отслеживания правильности работы алгоритма.
Сравнить результаты кодирования с равномерным кодированием (ASCII) на текстах разной длины и сделать выводы об эффективности алгоритма.

1.2 Используемый язык программирования:

C++ 23 вместе со стандартными библиотеками: STD, STL (Vector, Map, Stack, Algorithm)

2 Алгоритм Фано

2.1 Реализация алгоритма на псевдокоде

```
// Главная функция
Function main()
    regime := 0
    inputFile := ""
    encodedFile := "encoded.bin"
    decodedFile := "decoded.txt"
    treeFile := "tree.txt"

    While true do
        print("Выберите режим: 1 - кодировка, 2 - декодировка, 0 - выход: ")
        regime := readInput()

        If regime = 0 then
            break
        End if

        If regime = 1 then
            print("Введите имя файла для кодирования: ")
            inputFile := readInput()
            fanoEncoding(inputFile, encodedFile, treeFile)
            print("Файл закодирован и записан в " + encodedFile)
        Else If regime = 2 then
            print("Введите имя файла для декодирования: ")
            encodedFile := readInput()
            fanoDecoding(encodedFile, treeFile, decodedFile)
            print("Файл декодирован и записан в " + decodedFile)
        End if
    End while
End function
```

Фун. 1 Главная функция

```
// Вход: inputFile - файл с текстом, outputFile - файл для записи закодированных данных, t
Function fanoEncoding(inputFile, outputFile, treeFile)
    text := readFile(inputFile)

    // Получаем вероятности символов
    probabilitiesMap := getProbabilities(text)

    // Преобразуем в вектор пар
    probabilities := []
    For each (symbol, probability) in probabilitiesMap do
        probabilities.push((symbol, probability))
    End for

    // Строим дерево Фано
    root := buildFanoTree(probabilities)
```

```
    // Генерируем коды
    codes := пустой массив
    generateCodes(root, codes)

    // Кодируем текст
    encodedBytes := encode(text, codes)

    // Записываем закодированные байты в файл
    writeToFile(outputFile, encodedBytes)

    // Записываем коды в файл
    writeCodesToFile(codes, treeFile)
End function
```

Фун. 2 Основная функция кодирования

```
// Вход: text - строка текста
// Выход: probabilities - таблица вероятностей символов
Function getProbabilities(text)
    frequency := пустой массив
    For each c in text do
        frequency[c]++
    End for

    probabilities := пустой массив
    For each (symbol, freq) in frequency do
        probabilities[symbol] := freq / size(text)
    End for

    return probabilities
End function
```

Фун. 3 getprobabilities

```
// Вход: probabilities - массив пар (символ, вероятность)
// Выход: корень дерева Фано
Function buildFanoTree(probabilities)
    nodes := пустой массив

    // Создаем узлы для каждого символа
    For each (symbol, probability) in probabilities do
        nodes.push(Node(symbol, probability))
    End for

    // Пока остается более одного узла
    While size(nodes) > 1 do
        sort(nodes) // Сортируем по вероятностям

        // Объединяем два узла с наименьшими вероятностями
        left := nodes[0]
```

```
        right := nodes[1]
        parent := Node('\0', left.probability + right.probability)
        parent.left := left
        parent.right := right

        // Удаляем два узла и добавляем родительский
        nodes.erase(0)
        nodes.erase(0)
        nodes.push(parent)
    End while

    return nodes[0] // Корень дерева
End function
```

Фун. 4 Построение дерева Фано

```
// Вход: root - корень дерева Фано, codes - пустой массив для кодов
// Выход: заполненный массив codes с кодами для каждого символа
Function generateCodes(root, codes)
    If root = NULL then
        return
    End if

    stack := пустой стек
    push(stack, (root, "")) // Корень дерева и пустая строка для кода

    While stack не пустой do
        (node, code) := pop(stack)

        If node.symbol  '\0' then // Это лист дерева
            codes[node.symbol] := code
        End if

        If node.right  NULL then
            push(stack, (node.right, code + "1"))
        End if

        If node.left  NULL then
            push(stack, (node.left, code + "0"))
        End if
    End while
End function
```

Фун 5. Генерация кодов для символов

```
// Вход: text - строка текста, codes - таблица кодов символов
// Выход: массив байтов encodedBytes с закодированным текстом
Function encode(text, codes)
    encodedString := ""

    // Переводим текст в битовую строку
```

```
    For each c in text do
        encodedString += codes[c]
    End for

    encodedBytes := пустой массив

    // Записываем длину текста в первые 4 байта
    length := size(text)
    For i from 0 to 3 do
        encodedBytes.push((length >> (24 - 8 * i)) & 0xFF)
    End for

    // Записываем битовую строку в байты
    For i from 0 to size(encodedString) step 8 do
        byte := 0
        bitsToRead := min(8, size(encodedString) - i)

        For j from 0 to bitsToRead - 1 do
            byte := (byte << 1) | (encodedString[i + j] - '0')
        End for

        If bitsToRead < 8 then
            byte := byte << (8 - bitsToRead)
        End if

        encodedBytes.push(byte)
    End for

    return encodedBytes
End function
```

Фун 6. Кодирование текста

```
// Вход: encodedFile - файл с закодированными данными, treeFile - файл с кодами, outputFil
Function fanoDecoding(encodedFile, treeFile, outputFile)
    encodedBytes := readFile(encodedFile)

    // Получаем таблицу кодов из файла
    codes := пустой массив
    maxCodeLength := 0

    treeFileIn := openFile(treeFile)
    While not endOfFile(treeFileIn) do
        line := readLine(treeFileIn)
        If line contains "RETURN" then
            line.replace("RETURN", "\n")
        End if
        symbol := line[0]
        code := line[1..]
        codes[code] := symbol
        maxCodeLength := max(maxCodeLength, size(code))
```

```
    End while
    closeFile(treeFileIn)

    // Декодируем строку
    decodedString := decode(encodedBytes, codes)

    // Записываем декодированную строку в файл
    writeToFile(outputFile, decodedString)
End function
```

Фун 7. Основная функция декодирования

```
// Вход: encodedBytes - массив байтов, codes - таблица (код -> символ)
// Выход: строка decodedString
Function decode(encodedBytes, codes)
    decodedString := ""
    currentCode := ""

    // Читаем длину текста из первых 4 байтов
    targetLength := 0
    For i from 0 to 3 do
        targetLength := (targetLength << 8) | encodedBytes[i]
    End for

    decodedLength := 0
    bitCount := (size(encodedBytes) - 4) * 8 // Количество битов для декодирования

    // Проходим по всем битам, начиная с 5-го байта
    For i from 4 * 8 to bitCount + 4 * 8 - 1 do
        byteIndex := i // Индекс байта
        bitPosition := 7 - (i % 8) // Позиция бита в байте

        currentCode += (encodedBytes[byteIndex] >> bitPosition) & 1 ? "1" : "0"

        If codes.contains(currentCode) then
            decodedString += codes[currentCode]
            decodedLength += 1
            currentCode := ""
            If decodedLength = targetLength then
                break
            End if
        End if

        If size(currentCode) > 20 then // Максимальная длина кода
            currentCode := ""
        End if
    End for

    return decodedString
End function
```

Фун 8. Функция декодирования байтов

3 Пример работы алгоритма

Возьмём для примера строчку "abcabcdas"Для начала необходимо посчитать частоты появления символов в тексте

1. P(a) = 3/9 = 0.33
2. P(b) = 2/9 = 0.22
3. P(c) = 2/9 = 0.22
4. P(d) = 1/9 = 0.11
5. P(s) = 1/9 = 0.11

Символ	Вероятность Pi	Код	Длинна кода Li	Pi*Li
a	0.33	00	2	0.67
b	0.22	01	2	0.47
c	0.22	10	2	0.44
d	0.11	110	3	0.33
e	0.11	111	3	0.33

$$\sum_{i=1}^{n} P_i * L_i = 2.22$$

- средняя длинна одного кода.
Запишем закодированную строку:

```
000110000011011000111
```

Длинна закодированного текста - 12 бит. Длинна исходного текста - 72 бит. Для нашего случая размер текста был сжат в 6 раз (без учета места выделенного на хранение дерева кодов).

4 Область применения алгоритма Фано

4.1 Область применения

1. Сжатие данных: Алгоритм может использоваться для создания компактных представлений символов (например, в текстах или файлах), что помогает уменьшить объем передаваемых или хранимых данных.

2. Передача данных: Этот алгоритм полезен для передачи данных по каналам с ограниченной пропускной способностью, где важно минимизировать объем передаваемой информации.

3. Оптимизация кодирования: В случае, если известны вероятности появления символов, алгоритм Фано помогает построить оптимальный префиксный код для минимизации общей длины сообщения.

4.2 Возможные ошибки:

1. Неправильные вероятности символов: Если вероятности символов рассчитаны неправильно или не соответствуют реальному распределению символов, программа может построить неэффективные коды. Например, символы с высокой частотой могут получить слишком длинные коды, а редкие символы —- короткие. Это нарушает принцип оптимального кодирования.

2. Малое количество символов: Если в сообщении слишком мало символов, вероятность одинаковых символов будет высокой, и программа может создать неэффективные коды.

3. Крайние случаи вероятностей: Если все символы имеют почти одинаковую вероятность, алгоритм может не сжать данные эффективно, поскольку различие в длине кодов между символами будет минимальным.

Алгоритм Фано лучше всего работает для файлов с большим количеством повторяющихся символов и четко различимыми вероятностями. Однако сбои могут возникнуть при некорректной обработке данных, неправильных вероятностях, малом объеме данных или ошибках ввода-вывода файлов.

5 Сравнение результатов кодирования с равномерным кодированием (ASCII) на текстах разной длины.

Генерируем случайные тексты длинной 10, 100, 1000, 10000, 100000 байт и сравниваем результаты сжатия. Получаем следующую таблицу:

Длина текста (байт)	Размер после кодирования (байт)	Размер до кодирования (ASCII)	Коэффициент сжатия
10	32	80	2.50
100	466	800	1.72
1000	4769	8000	1.68
10000	47720	80000	1.68
100000	477087	800000	1.68

Выводы:

1. Алгоритм Фано показывает значительное сжатие данных, особенно на коротких текстах. Например, для текста длиной 10 байт коэффициент сжатия составил 2.5, что означает, что размер кода Фано почти в 2.5 раза меньше, чем размер ASCII.

2. С увеличением длины текста коэффициент сжатия уменьшается и стабилизируется примерно на уровне 1.68 для текстов длиной 1000 байт и более. Это может быть связано с тем, что с увеличением объема данных распределение символов становится более равномерным.

3. Алгоритм Фано, основываясь на частотах символов, создает более короткие коды для более часто используемых символов (по сравнению с ASCII), что в итоге приводит к меньшему размеру кодированного текста.