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

17 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)

1.3 Ограничения

Невозможно закодировать символ табуляции.
Невозможно раскодировать файл размером более 4,294,967,295 байт.

2 Алгоритм Фано

2.1 Псевдокод

```
// Вход: start, end - границы диапазона, nodes - массив узлов
// Выход: индекс разделения узлов
Function med(start, end, nodes)
    left_sum := 0.0
    For i from start to end do
        left_sum += nodes[i].probability
    End for

    right_sum := 0.0
    m := end
    While m > start and abs(left_sum - right_sum) >= abs(nodes[m].probability) do
        left_sum -= nodes[m].probability
        right_sum += nodes[m].probability
        m -= 1
    End while

    return m
End function
```

Фун. 1. Функция нахождения индекса разделения узлов

```
// Вход: nodes - массив узлов
// Выход: корень дерева Фано
Function buildFanoIterative(nodes)
    stack := пустой стек
    totalProb := 0.0

    For each node in nodes do
        totalProb += node.probability
    End for

    root := Node('\0', totalProb)
    push(stack, (0, size(nodes) - 1, root))

    While not stack.empty() do
        (start, end, parent) := pop(stack)

        If start = end then
            parent.left := nodes[start]
            continue
        End if

        split := med(start, end, nodes)

        leftChild := Node('\0', 0.0)
        rightChild := Node('\0', 0.0)
```

```
        For i from start to split do
            leftChild.probability += nodes[i].probability
        End for
        For i from split + 1 to end do
            rightChild.probability += nodes[i].probability
        End for

        parent.left := leftChild
        parent.right := rightChild

        push(stack, (start, split, leftChild))
        push(stack, (split + 1, end, rightChild))
    End while

    return root
End function
```

Фун. 2. Итеративное построение дерева Фано

```
// Вход: root - корень дерева Фано, codes - таблица 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, ""))

    While not stack.empty() do
        (node, code) := pop(stack)

        If node.symbol  '\0' then
            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. 3. Генерация кодов для символов

```
// Вход: codes - таблица кодов, treeFile - имя файла
Function writeCodesToFile(codes, treeFile)
```

```
    outTreeFile := openFile(treeFile, ios::trunc)
    If outTreeFile = NULL then
        print("Ошибка при открытии файла для записи кодов.")
        return
    End if

    For each (symbol, code) in codes do
        If code  "" then
            If symbol = '\n' then
                outTreeFile.write("RETURN" + "\t" + code + "\n")
            Else
                outTreeFile.write(symbol + "\t" + code + "\n")
            End if
        End if
    End for

    outTreeFile.close()
End function
```

Фун. 4. Запись кодов в файл

```
// Вход: text - строка текста, codes - таблица кодов символов
// Выход: массив байтов encodedBytes с закодированным текстом
Function encode(text, codes)
    encodedBytes := пустой массив
    encodedString := ""
    length := size(text) - 1

    For each c in text do
        If codes.contains(c) then
            encodedString += codes[c]
        End if
    End for

    For i from 0 to 3 do
        encodedBytes.push((length >> (24 - 8 * i)) & 0xFF)
    End for

    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. 5 Кодирование текста

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

    targetLength := 0
    For i from 0 to 3 do
        targetLength := (targetLength << 8) | encodedBytes[i]
    End for

    decodedLength := 0
    bitCount := (size(encodedBytes) - 4) * 8

    For i from 4 * 8 to bitCount + 4 * 8 - 1 do
        byteIndex := i
        bitPosition := 7 - (i % 8)

        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
            currentCode := ""
        End if
    End for

    If size(currentCode) > 0 and size(currentCode) <= 20 then
        If codes.contains(currentCode) then
            decodedString += codes[currentCode]
        End if
    End if

    return decodedString
End function
```

Фун. 6. Функция декодирования байтов

```
// 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
```

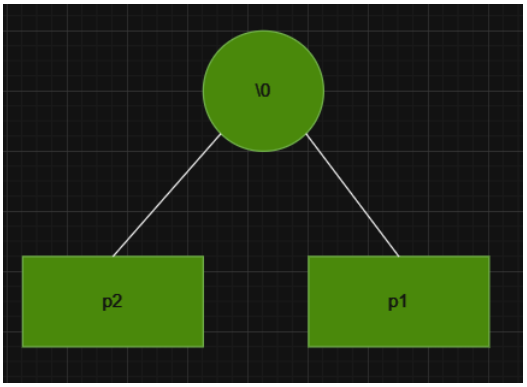
Фун. 7. Получение вероятностей символов

3 Пример работы алгоритма

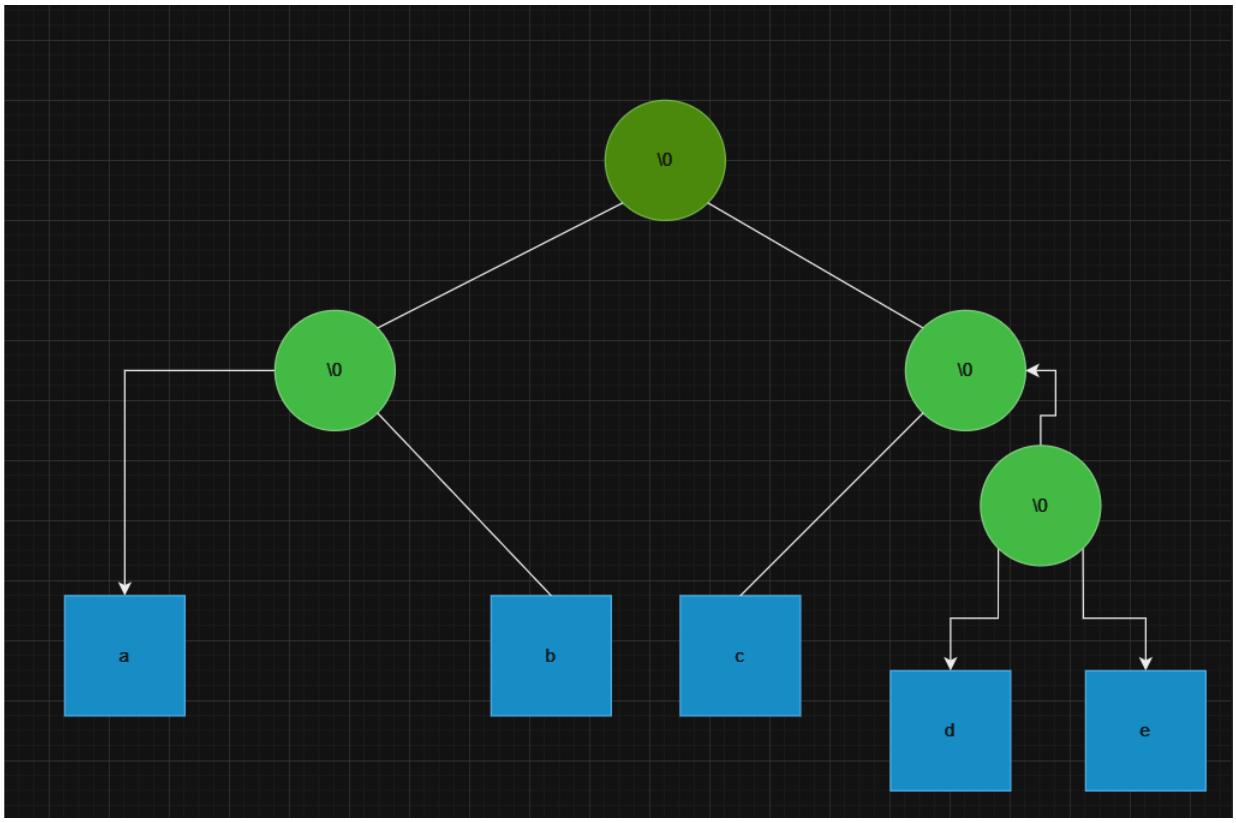
Возьмём для примера строчку "abcabcdas"Для начала необходимо посчитать частоты появления символов в тексте

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

Для начала строит дерево Фано: Строим массив где записываем вероятности по убыванию: $p := [0.11, 0.11, 0.22, 0.22, 0.33]$
Далее делим массив на два равновероятных массива:
$med(1, 5, p)$
Получаем: $p_1 := [0.11, 0.11, 0.22]$ и $p_2 := [0.22, 0.33]$.
Получаем дерево, где в родительском узле символ окончания строки, в левом - p_2, в правом - p_1. Отправляем оказатель на родитель в стек.



Проделываем заново операцию с p_1 и p_2.
$med(1, 3, p_1)$, $med(1, 2, p_2)$.
Получаем $p_3 := [0.11, 0.11]$, $p_4 := [0.22]$, $p_5 := [0.22]$, $p_6 := [0.33]$.
Строим два дерва, аналогично первому шагу, и отправляем два родительских узла в стек. Сначала дерево, с наименьшей суммой вероятностей.
Далее, по новой: $med(1, 2, p_3)$, $p_7 := [0.11], p_8 := [0.11]$, получаем очередное дерево и отправляем в стек.
Доставая поочередно узлы из стека, реконструируем дерево. В итоге получается дерево, листья которых содержат символы упорядоченные справа налево в порядке убывания вероятности.



Делая обход в ширину справа-налево, можно назначить каждому символу код.
Получается следующая таблица:

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.