Циклический код. Построение оптимальных кодов алгоритма Хаффмена.

Содержание

Введение

Как положительную тенденцию, отвечающую требованиям сегодняшнего дня, следует отметить ускоренный рост услуг связи и информатизации, которые за год возросли на 41,6 процента. Это обеспечено в первую очередь за счет увеличения количества абонентов, пользующихся услугами мобильной связи и сети Интернет, чему способствовало принятие мер в отчетном году по снижению тарифов для населения на услуги по предоставлению доступа в Интернет на 22 процента. Сегодня около 8 миллионов человек являются активными пользователями сети Интернет.

В истекшем году большое внимание уделялось проведению активной инвестиционной политики, направленной на ускорение модернизации, технического и технологического перевооружения действующих и создание новых, современных, высокотехнологичных производств [1].

Циклические коды составляют большую группу наиболее широко используемых на практике линейных, систематических кодов. Их основное свойство, давшее им название, состоит в том, что каждый вектор, получаемый из исходного кодового вектора путем циклической перестановки его символов, также является разрешенным кодовым вектором.

Операции кодирования и декодирования ЦК сводятся к известным процедурам умножения и деления полиномов. Для двоичных кодов эти операции легко реализуются технически с помощью линейных переключательных схем (ЛПС), при этом получаются относительно простые схемы кодеков, в чем состоит одно из практических достоинств ЦК.[9]

Алгоритм Хаффмана основывается на том, что символы в текстах как правило встречаются с различной частотой. При обычном кодировании мы каждый символ записываем в фиксированном количестве бит, например каждый символ в одном байте, или в двух. Однако, т. к. некоторые символы встречаются чаще, а некоторые реже − можно записать часто встречающиеся символы в небольшом количестве бит, а для редко встречающихся символов использовать более длинные коды. Тогда суммарная длина закодированного текста может стать меньше.[2]

Сжатие данных по Хаффману применяется при сжатии фото- и видеоизображений (JPEG, стандарты сжатия MPEG), в архиваторах (PKZIP, LZH и др.), в протоколах передачи данных MNP5 и MNP7.[7]

Задача: Построение алгоритма и разработка программы вычисление кода Хаффмана.

1. Циклический код

Циклический код — линейный код, обладающий свойством цикличности, то есть каждая циклическая перестановка кодового слова также является кодовым словом. Используется для преобразования информации для защиты её от ошибок (см. Обнаружение и исправление ошибок).[8]

Принято описывать циклические коды (ЦК) при помощи порождающих полиномов G(X) степени m = n – k, где m– число проверочных символов в кодовом слове. В связи с этим ЦК относятся к разновидности полиномиальных кодов.[9]

Общие понятия и определения.

Широкое распространение на практике получил класс линейных кодов, которые называются циклическими. Данное название происходит от основного свойства этих кодов, а именно, если некоторая кодовая комбинация принадлежит циклическому коду, то комбинация, полученная циклической перестановкой исходной комбинации (циклическим сдвигом).[10]

Любой групповой код (n,k)может быть записан в виде матрицы, включающей k линейно независимых строк по *n* символов и, наоборот, любая совокупность k линейно независимых n-разрядных кодовых комбинаций может рассматриваться как образующая матрица некоторого группового кода. Среди всего многообразия таких кодов можно выделить коды, у которых строки образующих матриц связаны дополнительным условием цикличности.[4]

Пусть $\overline{y} = (y_0, y_1, \ldots, y_{n-1}) \in \mathbb{GF}(q^n)$ слово длины n над алфавитом из элементов конечного поля $\mathbb{GF}(q)$ и $y(x) = y_0 + y_1 x + y_2 x^2 + \ldots + y_{n-1} x^{n-1}$ полином, соответствующий этому слову, от формальной переменной *x*. Видно, что это соответствие не просто взаимно однозначное, но и изоморфное. Так как «слова» состоят из букв из поля, то их можно складывать и умножать (поэлементно), причём результат будет в том же поле. Полином, соответствующий линейной комбинации $\overline{y} = m_1\overline{y_1} + m_2\overline{y_2}$ пары слов

$\overline{y_1} = (y_{1,0}, \ldots, y_{1,n-1})$ и $\overline{y_2}$ и $\overline{y_2} = (y_{2,0}, \ldots, y_{2,n-1})$, равен линейной комбинации полиномов этих слов

$$\overline{y}(x) = \sum_{k=0}^{n-1}(m_1 y_{1k} + m_2 y_{2k})x^k = m_1\overline{y_1}(x) + m_2\overline{y_2}(x)$$

Это позволяет рассматривать множество слов длины n над конечным полем как линейное пространство полиномов со степенью не выше *n-1* над полем $\mathbb{GF}(q)$

Алгебраическое описание

Если $\overrightarrow{c_1}$ кодовое слово, получающееся циклическим сдвигом на один разряд влево из слова $\overrightarrow{c}$, то соответствующий ему полином $c_1(x)$ получается из предыдущего умножением на x:

$c_1(x) = xc(x) \mod (x^n - 1)$, пользуясь тем, что $x^n \equiv 1 \mod (x^n - 1)$,

Сдвиг вправо и влево соответственно на разрядов:

$$c_j(x) = x^j c(x) \mod (x^n - 1)$$
$$c_{-j}(x)x^j = c(x) \mod (x^n - 1)$$

Если *m(x)* — произвольный полином над полем *GF(q)* и *c(x)* — кодовое слово циклического *(n,k)* кода, то *m(x)c(x) mod(x^n-1)*тоже кодовое слово этого кода. [7]

Кодирование

- **Несистематическое**

При несистематическом кодировании кодовое слово получается в виде произведения информационного полинома на порождающий

c(x)=m(x)g(x)

Оно может быть реализовано при помощи перемножения полиномов.

- **Систематическое**

При систематическом кодировании кодовое слово формируется в виде информационного подблока и проверочного

$$c(x) = [s(x)\ m(x)]$$

Пусть информационное слово образует старшие степени кодового слова, тогда

$$c(x) = x^r m(x) + s(x), r = n - k$$

Тогда из условия, $c(x) = x^r m(x) + s(x) = 0 \mod g(x)$ следует $s(x) = -x^r m(x) \mod g(x)$

Это уравнение и задает правило систематического кодирования. Оно может быть реализовано при помощи многотактных линейных фильтров(МЛФ)[3]

Пример

Дано: Двоичный (7,4,3) код

В качестве делителя x^7-1 выберем порождающий полином третьей степени $g(x)=x^3+x+1$, тогда полученный код будет иметь длину $n=7$, число проверочных символов (степень порождающего полинома) $r=3$, число информационных символов $k=4$, минимальное расстояние $d=3$.

Порождающая матрица кода:

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix},$$

где первая строка представляет собой запись полинома $g(x)$ коэффициентами по возрастанию степени. Остальные строки — циклические сдвиги первой строки.

Проверочная матрица:

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix},$$

где *i*-ый столбец, начиная с *1-ого*, представляет собой остаток от деления x^i на полином *g(x)*, записанный по возрастанию степеней, начиная сверху.

Так, например, 4-ий столбец получается $h_3(x) = x^3 \mod g(x) = x + 1$, или в векторной записи [12].

Легко убедиться, что $GH^T=0$.

2. Алгоритм Хаффмана

Алгоритм Хаффмана — адаптивный простой алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.[6]

Один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности 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 (т.е. 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 (Н-дерево). [11]

Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.

В отличие от алгоритма Шеннона — Фано, алгоритм Хаффмана остаётся всегда оптимальным и для вторичных алфавитов m2 с более чем двумя символами.

Алгоритм Хаффмана основывается на том, что символы в текстах как правило встречаются с различной частотой. При обычном кодировании мы каждый символ записываем в фиксированном количестве бит, например каждый символ в одном байте, или в двух.

Однако, т. к. некоторые символы встречаются чаще, а некоторые реже − можно записать часто встречающиеся символы в небольшом количестве бит, а для редко встречающихся символов использовать более длинные коды. Тогда суммарная длина закодированного текста может стать меньше.[13]

Этот метод кодирования состоит из двух основных этапов:

1. Построение оптимального кодового дерева.

2. Построение отображения код-символ на основе построенного дерева.

Классический алгоритм Хаффмана имеет ряд существенных недостатков. Во-первых, для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, которой пользовался кодер. Следовательно, длина сжатого сообщения увеличивается на длину таблицы частот, которая должна посылаться впереди данных, что может свести на нет все усилия по сжатию сообщения. Кроме того, необходимость наличия полной частотной статистики перед началом собственно кодирования требует двух проходов по сообщению: одного для построения модели сообщения (таблицы частот и Н-дерева), другого для собственно кодирования. Во-вторых, избыточность кодирования обращается в ноль лишь в тех случаях, когда вероятности кодируемых символов являются обратными степенями числа 2. В-третьих, для источника с энтропией, не превышающей 1 (например, для двоичного источника), непосредственное применение кода Хаффмана бессмысленно.[13]

Виды алгоритма Хаффмана

1) статический

2) динамический

3) адаптивный

Рассмотренный пример относится к динамическому алгоритму Хаффмана.

Статический алгоритм отличается от него тем, что таблица частот не вычисляется по тексту, а используется некоторая ранее вычисленная таблица. Соответственно, для динамического алгоритма требуется два прохода по тексту, первый для построения таблицы, второй − для кодирования. Для статического алгоритма требуется только один проход, т. к. таблица уже есть.

Естественно, при использовании статического алгоритма возникает риск того, что алгоритм будет не эффективен из-за несоответствия реальных частот и значений в таблице.

Тем не менее иногда статический алгоритм можно применять, например, известно, что в обычных текстах, например художественная литература,

журналы, частоты символов как правило примерно одинаковые для различных текстов.

Адаптивный алгоритм мы не рассматриваем здесь. Вкратце можно сказать, что в этом алгоритме также используется один проход, но при этом дерево постоянно перестраивается, т. е. меняются коды символов.[5]

3. Алгоритм построение кода Хаффмана

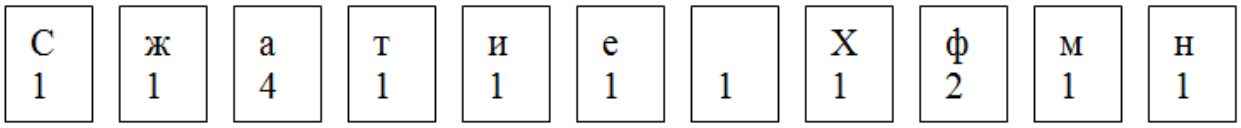
Закодируем строку "Сжатие Хаффмана"

Вначале нужно подсчитать количество вхождений каждого символа в тексте.

С	ж	А	т	И	е		Х	ф	м	н
1	1	4	1	1	1	1	1	2	1	1

Эта таблица называется "таблица частот". Теперь будем строить дерево. Узел дерева будет образован набором символов и числом - суммарным количеством вхождений данных символов в тексте. Назовем указанное число - весом узла.

Листьями дерева будут узлы, образованные одним символом:

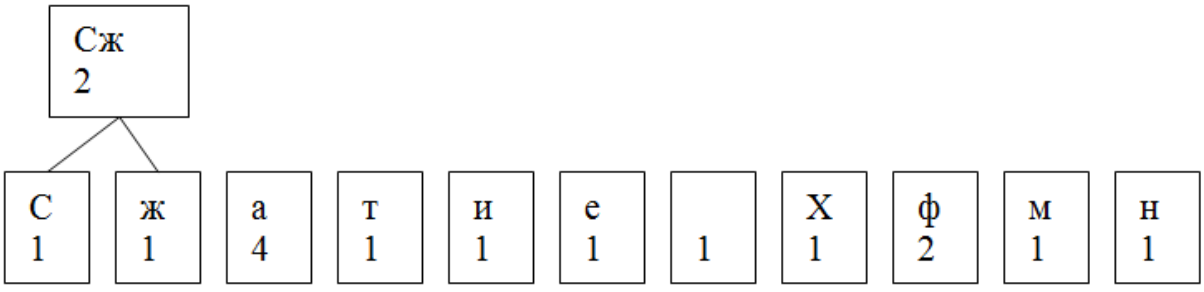


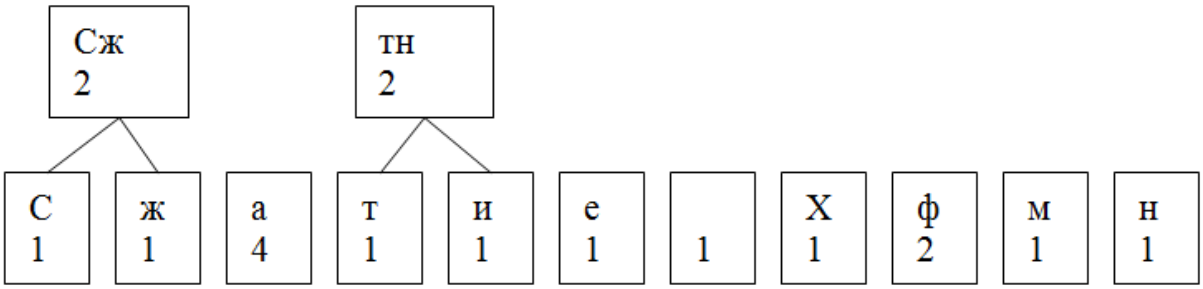
Теперь будем циклически делать следующее:

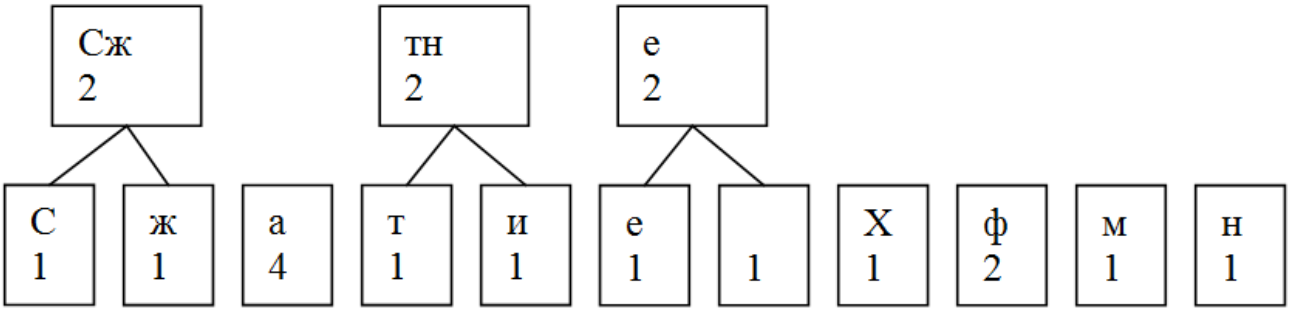
1) Ищем среди узлов, не имеющих родителя, два узла, имеющих в сумме наименьший вес.

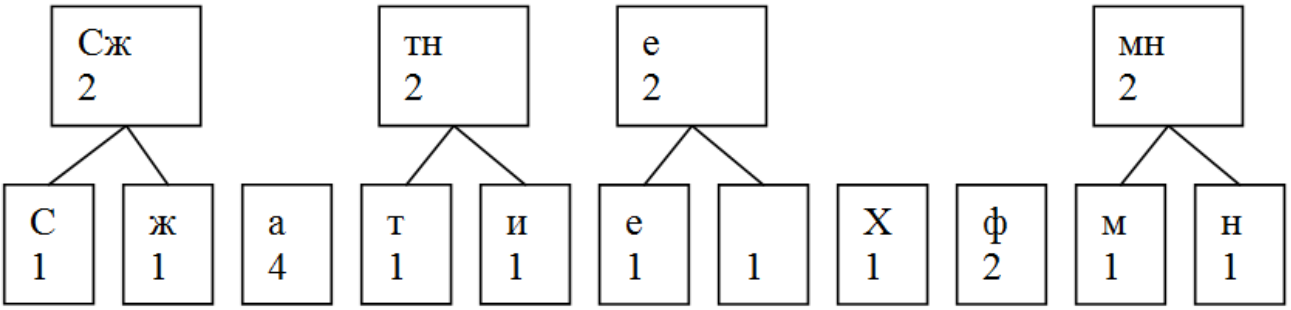
2) Создаем новый узел, его потомками будут два выбранных узла. Его весом будет сумма весов выбранных узлов. Его набор символов будет образован в результате объединения наборов символов выбранных узлов.

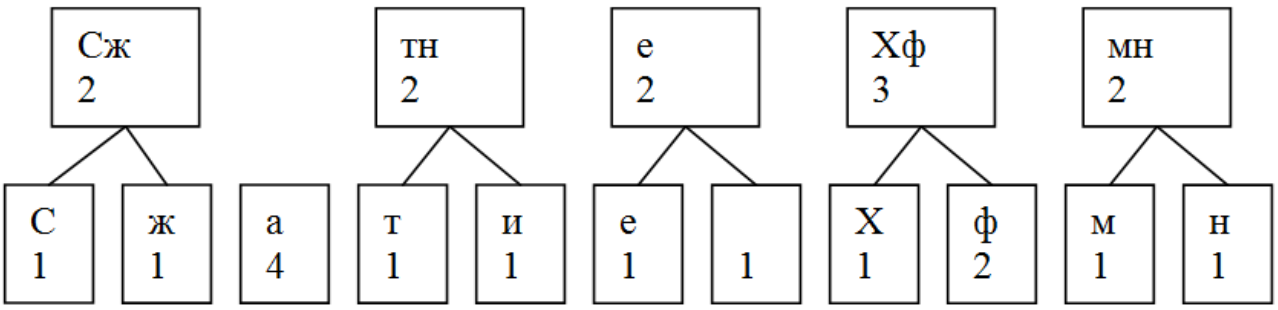
Создадим узлы для «лепестков»

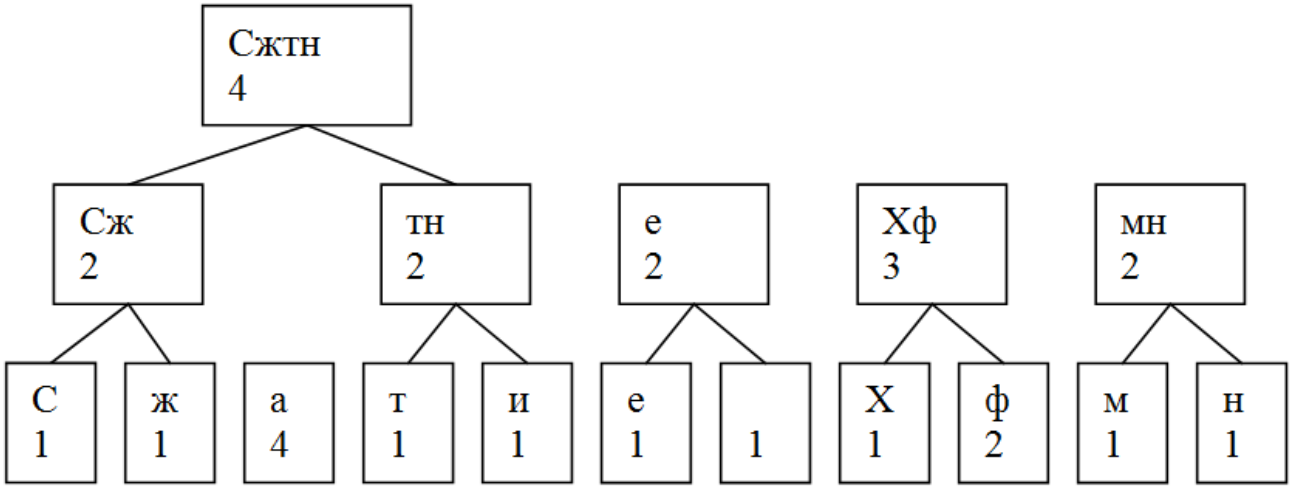


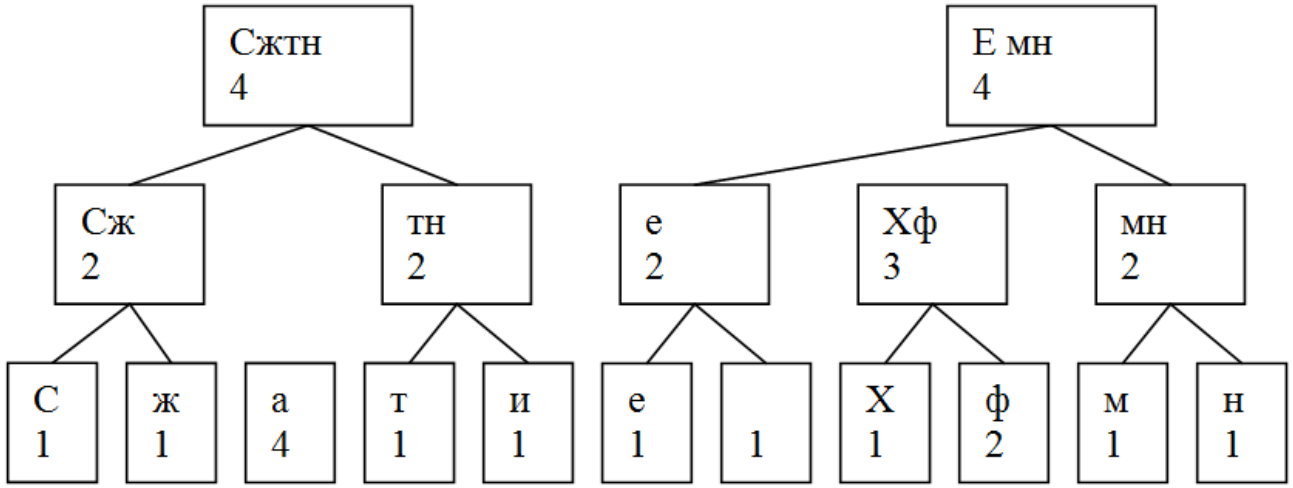

Сж
2
тн
2
С
1
ж
1
а
4
т
1
и
1
е
1
1
Х
1
ф
2
м
1
н
1

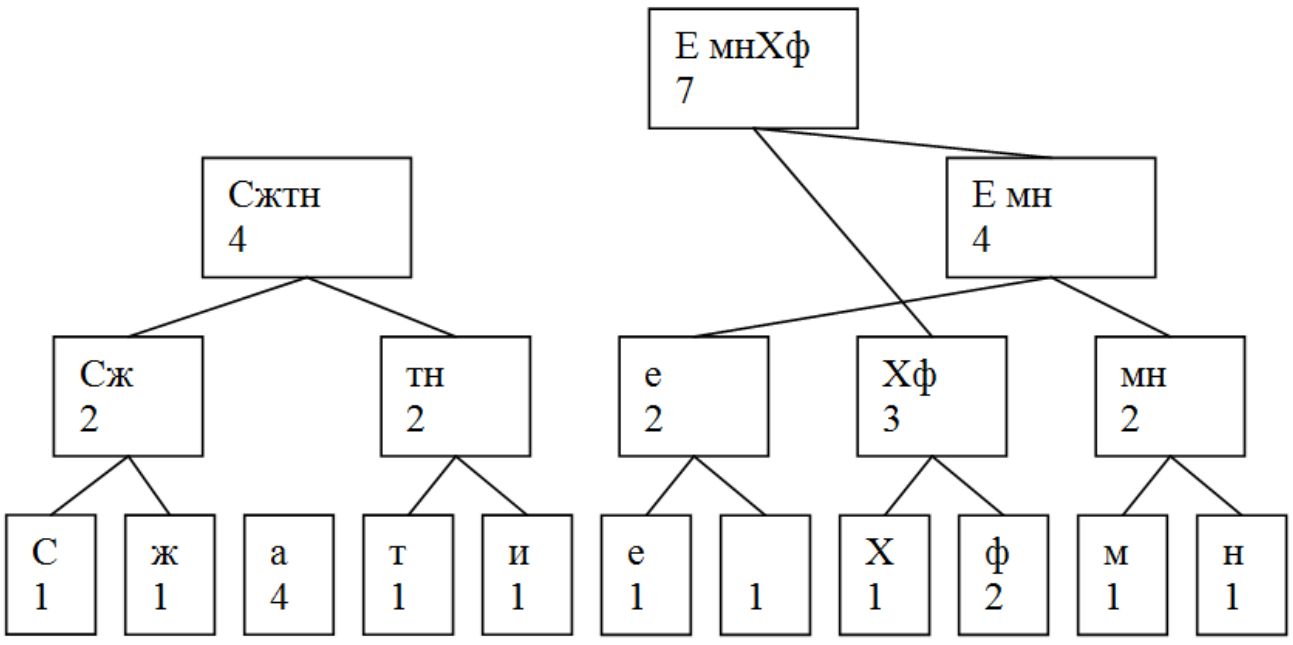

Сж
2
тн
2
е
2
С
1
ж
1
а
4
т
1
и
1
е
1
1
Х
1
ф
2
м
1
н
1

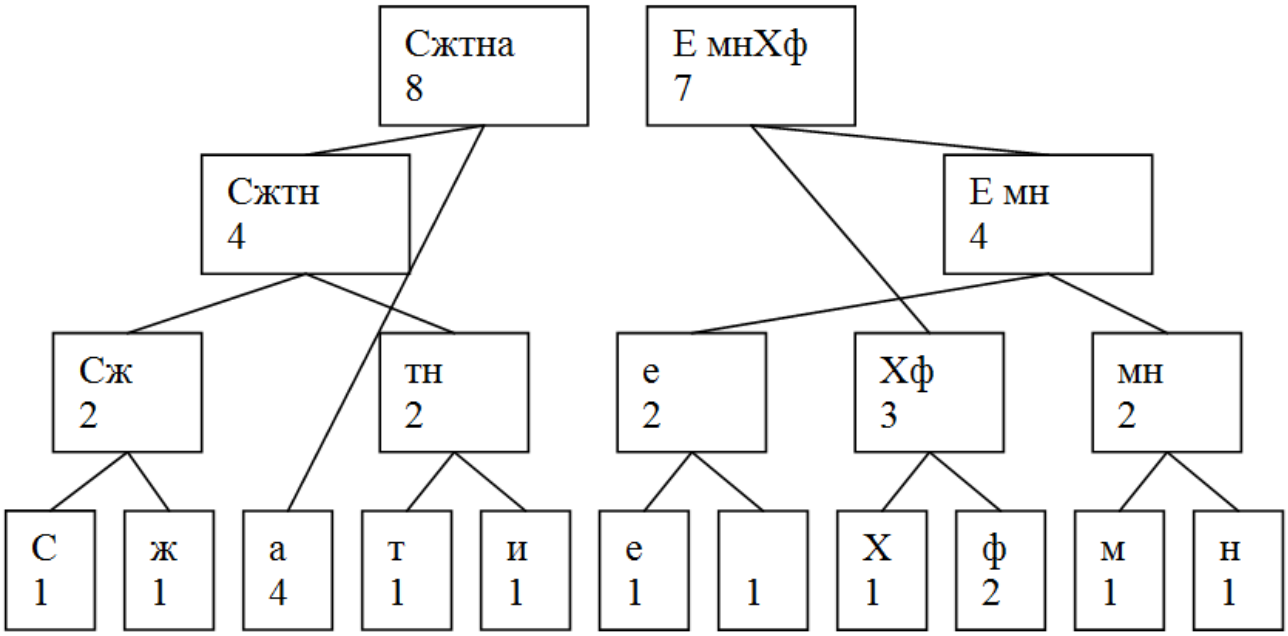

Сж
2
тн
2
е
2
мн
2
С
1
ж
1
а
4
т
1
и
1
е
1
1
Х
1
ф
2
м
1
н
1

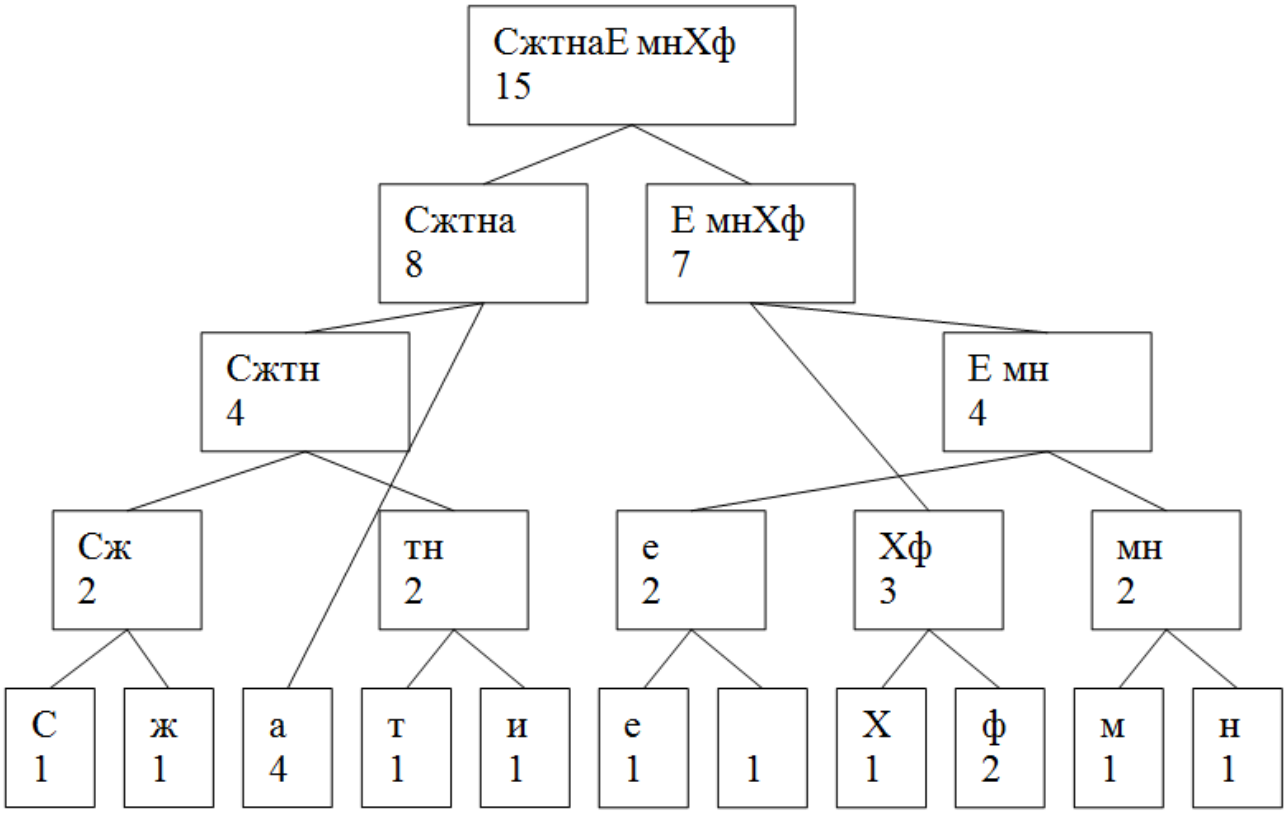

Сж
2
тн
2
е
2
Хф
3
мн
2
С
1
ж
1
а
4
т
1
и
1
е
1
1
Х
1
ф
2
м
1
н
1


Сжтн
4
Сж
2
тн
2
е
2
Хф
3
мн
2
С
1
ж
1
а
4
т
1
и
1
е
1
1
Х
1
ф
2
м
1
н
1

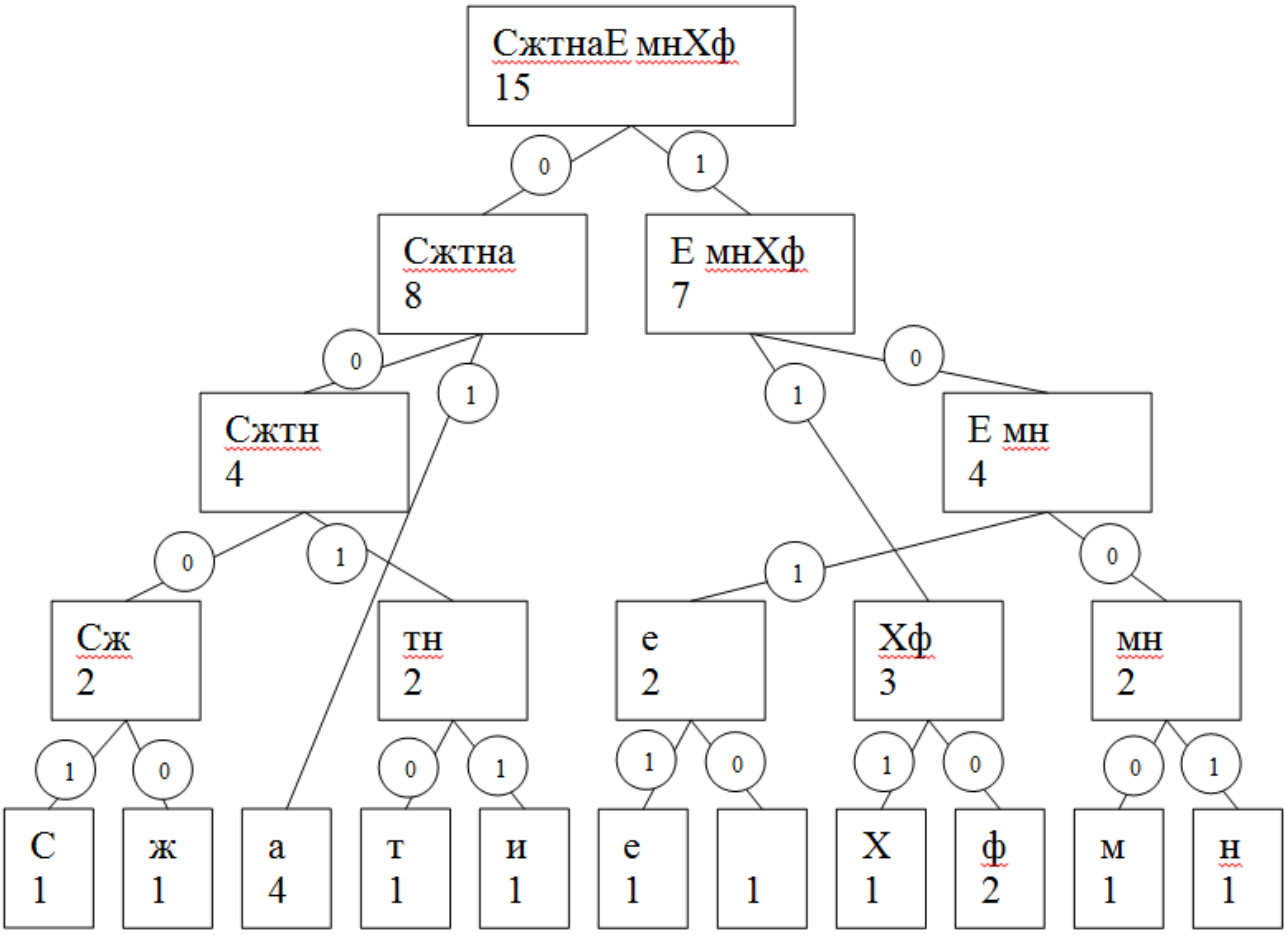

Сжтн
4
Е мн
4
Сж
2
тн
2
е
2
Хф
3
мн
2
С
1
ж
1
а
4
т
1
и
1
е
1
1
Х
1
ф
2
м
1
н
1


Е мнХф
7
Сжтн
4
Е мн
4
Сж
2
тн
2
е
2
Хф
3
мн
2
С
1
ж
1
а
4
т
1
и
1
е
1
1
Х
1
ф
2
м
1
н
1





Теперь для каждого узла, имеющего потомков, пометим дуги, идущие вниз, на одной дуге напишем 0, на другой 1.



Теперь, чтобы получить код символа, надо, начиная с корня дерева идти вниз до листа соответствующего данному символу. При этом следует записывать те значения, которые написаны на тех дугах, которые мы проходим.

Получаются следующие коды

С	0001
Ж	0000
А	01
Т	0010
И	0011
Е	1011
	1010
Х	111
Ф	110
М	1000
Н	1001

Заметим, что код является префиксным, т. е. ни один код символа не является префиксом кода другого символа. Также видно, что для часто встречающихся символов коды короче.

Как будет выглядеть закодированная строка:

С	ж	а	т	и	е		Х	а	ф	ф	м	а	н	а
0001	0000	01	0010	0011	1011	1010	111	01	110	110	1000	01	1001	01

Т. е.

000100000100100011101110101110111011010000110 0101

Декодирование

Для декодирования можно воспользоваться построенным деревом. Начинаем с корня дерева и в качестве текущего бита берем начало текста.

В цикле делаем следующее

1) Смотрим, какое значение у текущего бита, и идем в низ по дереву по дуге, на которой указано такое же значение.

2) Переходим к следующему биту.

3) Если сейчас находимся в листе дерева: читаем символ находящийся в этом листе и записываем его в результат декодирования, переходим снова в корень дерева.

Применение

Сжатие данных по Хаффману применяется при сжатии фото- и видеоизображений (JPEG, стандарты сжатия MPEG), в архиваторах (PKZIP, LZH и др.), в протоколах передачи данных MNP5 и MNP7.

Заключение

В данной курсовой работе было рассмотрено основы понятие циклического кода и алгоритм Хаффмана, а также теория построение и алгоритм кода Хаффмана. В работе изучено основные понятие создание циклического кода и построение алгоритма кода Хаффмана. Практической части курсовой работы было создано программа вычисление кода Хаффмана в языке программирование C++, а также разработан алгоритм программы.

Циклические коды — это целое семейство помехоустойчивых кодов, включающее в себя в качестве одной из разновидностей коды Хэмминга, но в целом обеспечивающее большую гибкость с точки зрения возможности реализации кодов с необходимой способностью обнаружения и исправления ошибок, возникающих при передаче кодовых комбинаций по каналу связи. Циклический код относится к систематическим блочным (n, k)-кодам, в которых k первых разрядов представляют собой комбинацию первичного кода, а последующие (n − k) разрядов являются проверочными.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (H-дерево). Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.

Литература

1. Доклад Президента Республики Узбекистан Ислама Каримова на заседании Кабинета Министров, посвященном основным итогам 2011 года и приоритетным направлениям социально-экономического развития Узбекистана на 2012 год.
2. Вернер М. Основы кодирования. — М.: Техносфера, 2004.
3. Питерсон В., Уэлдон Ф. Коды, исправляющие ошибки. — М.: Мир, 1976.
4. “Кодирование информации (двоичные коды)”.Березюк Н.Т., Андрущенко А.Г., Мощицкий С.С. и др. Харьков,издательское объединение “Вища школа”,1978. 252 с.
5. М.Н. Аршинов, Л.Е. Садовский, Коды и математика. 1983
6. ru.wikipedia.org/w/index.php?title=Алгоритм_Хаффмана
7. ru.wikipedia.org/w/index.php?title=Циклический_код
8. http://monetcom.eu/joomla/webcontent/courses/ISTU/IS/IS_Lec3_ru.pdf
9. http://informkod.narod.ru/5_5item.htm
10. http://yourtutor.narod.ru/cyclic/CyclicCodes.htm
11. http://dvo.sut.ru/libr/opds/i285vino/2.htm
12. http://book.itep.ru/2/28/corec_28.htm
13. http://stor.boom.ru/uch/din/1/15/153.htm

Приложение

Код программы построение циклического кода

```
#include <vector>
#include <string>
#include <climits>
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

int main ()
{
    char s[100], x[100], k[100];
    int l, S, d;
    cout<<"Informatsion simvollarni kiriting = ";
    cin>>s;
    l= strlen (s);
    int a = atoi(s);

    switch(l)
      {
           case 1:
                {
                if (l == 1)
                d=3;
                 }
           break;

           case 2:
                {
                if (l == 2)
                d=7;
                }
           break;

           case 3:
                {
                if (l == 3)
                d=11;
                }
           break;

           case 4:
                {
                if (l == 4)
                d=13;
                }
           break;
```

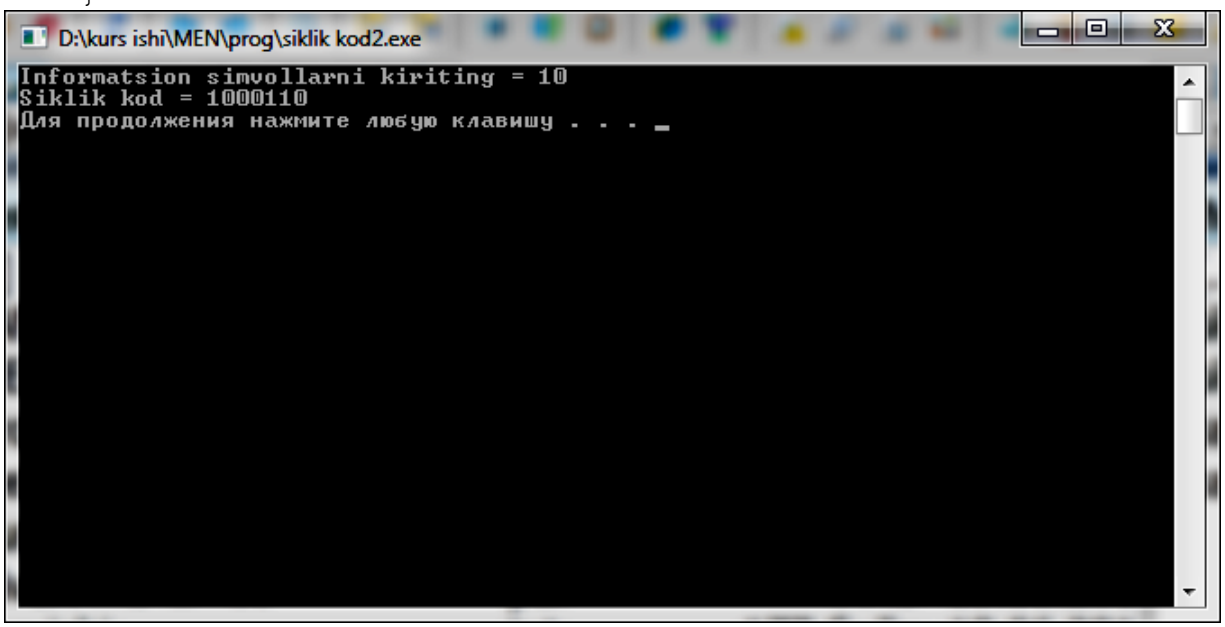
```
            case 5:
                  {
                  if (l == 5)
                  d=19;
                  }
             break;

            case 6:
                  {
                  if (l == 6)
                  d=25;
                  }
             break;

            case 7:
                  {
                  if (l == 7)
                  d=31;
                  }
             break;

              case 8:
                  {
                  if (l == 8)
                  d=37;
                  }
              break;
            }
          S=a*d;
          itoa(S, k, 2);
          cout<<"Siklik kod = "<<k<<endl;

      system("pause");
      return 0;
}
```



Код программы построение кода Хаффмана

```
#include <iostream>
#include <vector>
#include <string>
#include <climits>

using namespace std;
struct node
{
    node * leftChild;
    node * rightChild;
    double frequency;
    string content;
    string code;
 };

vector<node> nodeArray;

node  extractMin()
{
    double temp = (double) INT_MAX;
    vector<node>::iterator i1,pos;
    for(i1 = nodeArray.begin(); i1!=nodeArray.end(); i1++)
    {

        if(temp>(*i1).frequency)
        {
            pos = i1;
            temp = (*i1).frequency;
        }
    }

    node tempNode = (*pos);
    nodeArray.erase(pos);

    return tempNode;
}
node getHuffmanTree()
{
    while(!nodeArray.empty())
    {
        node * tempNode = new node;
        node * tempNode1 = new node;
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();


        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency+tempNode2-
>frequency;
```

```
            nodeArray.push_back(*tempNode);
            if(nodeArray.size() == 1)//only the root node exsits
            {
                break;
            }
        }
        return nodeArray[0];
    }

    void BFS(node * temproot,string s)
    {
        node * root1 = new node;
        root1 = temproot;

        root1->code = s;
        if(root1 == NULL)
        {

        }
        else if(root1->leftChild == NULL && root1->rightChild ==
NULL)
        {

            cout<<"киритилган элемент "<<root1->content<<endl;
            cout<<"ва унинг тузатувчи коди "<<root1->code<<endl;
        }
        else
        {
            root1->leftChild->code = s.append("0");
            s.erase(s.end()-1);
            root1->rightChild->code = s.append("1");
            s.erase(s.end()-1);


            BFS(root1->leftChild,s.append("0"));
            s.erase(s.end()-1);
            BFS(root1->rightChild,s.append("1"));
            s.erase(s.end()-1);
        }

    }

    void getHuffmanCode()
    {
        int size,i;
        double tempDouble;
        string tempString = "";

        cout<<"Кодланиши зарур бўлган маълумотнинг элементларнинг
туринининг сонини киритинг:"<<endl;
        cin>>size;

        for(i = 0; i<size; i++)
```

```
        {
            cout<<"Аввал код элементини ва унинг кодда учраш
сонини киритинг:"<<endl;
            node tempNode;
            cin>>tempString;
            cin>>tempDouble;


            tempNode.frequency = tempDouble;
            tempNode.content = tempString;
            tempNode.leftChild = NULL;
            tempNode.rightChild = NULL;
            nodeArray.push_back(tempNode);
        }

        node root = getHuffmanTree();

        BFS(&root,"");

    }

     int main()
    {
        vector<int> test;
        test.push_back(1);
        test.push_back(2);
        test.push_back(3);
        test.push_back(4);
        vector<int>::iterator i1 = test.begin();
        test.erase(i1);

        setlocale(LC_ALL,"Russian");
        getHuffmanCode();
        system("pause");
        return 0;
    }
```