MA’LUMOTLARNI SIQISHDA XOFMAN KODLASH TIZIMINING AHAMIYATI

Farmonov Sherzodbek Raxmonjonovich
Farg’ona davlat universiteti amaliy matematika va informatika kafedrasi katta o’qituvchisi
e-mail: farmonovsh@gmail.com
Mo’minov Muhammadali Hadyatillo o’g’li
Farg’ona davlat universiteti talabasi
e-mail: mnabijonov584@gmail.com

Annotatsiya. *Xofman kodlash tizimi (Huffman coding) ma'lum bir ma'lumotni siqish va kompressiya qilishda keng qo'llaniladigan algoritm bo'lib, avvalambor, ma'lumotlarni optimal tarzda kodlashni ta'minlaydi. U, ayniqsa, axborot nazariyasida va kompyuter ilmida samarali va optimal kodlash usuli sifatida e'tirof etilgan. Xofman kodlash usuli, ma'lumotlar to'plamidagi har bir belgining tasodifiy paydo bo'lish ehtimollariga asoslangan holda, qisqa kodlarni tez-tez uchraydigan belgilar uchun, uzoq kodlarni esa kamroq uchraydigan belgilar uchun ajratadi. Bu esa, siqishning samaradorligini oshiradi.*

Kalit so'zlar: *Xofman kodlash, Ma'lumot siqilishi, Kodlash algoritmi, Optimal kodlash, Kompressiya, Ehtimollik, Prefix kodlash, Kod daraxti, Raqamli axborot, Dasturlash, Kompyuter ilm-fani, Telekommunikatsiya tizimlari, Audio va video siqish, Huffman tree, Siqish samaradorligi, Ma'lumotlar siqish formatlari,GZIP, ZIP, Raqamli multimedia, Xatolikni tuzatish kodlari.*

Абстрактный. *Система кодирования Хаффмана представляет собой широко используемый алгоритм сжатия и сжатия определенной информации и, прежде всего, обеспечивает оптимальное кодирование информации. Он признан эффективным и оптимальным методом кодирования, особенно в теории информации и информатике. Метод кодирования Хофмана распределяет короткие коды для частых символов и длинные коды для менее частых символов на основе вероятностей случайного появления каждого символа в наборе данных. Это увеличивает эффективность сжатия.*

Ключевые слова: *Кодирование Хаффмана, Сжатие информации, Алгоритм кодирования, Оптимальное кодирование, Сжатие, Вероятность, Префиксное кодирование, Кодовое дерево, Цифровая информация, Программирование, Информатика, Телекоммуникационные системы, Сжатие аудио и видео, Дерево Хаффмана, Эффективность сжатия, Форматы сжатия данных, GZIP, ZIP, Цифровое мультимедиа, Коды с*

исправлением ошибок.

Abstract. *The Huffman coding system (Huffman coding) is an algorithm that is widely used in the compression and compression of certain information, and first of all, it provides optimal encoding of information. It is recognized as an efficient and optimal coding method, especially in information theory and computer science. The Hofmann coding method allocates short codes for frequent characters and long codes for less frequent characters based on the random occurrence probabilities of each character in the data set. This increases the efficiency of compression.*

Keywords: *Huffman coding, Information compression, Coding algorithm, Optimal coding, Compression, Probability, Prefix coding, Code tree, Digital information, Programming, Computer science, Telecommunication systems, Audio and video compression, Huffman tree, Compression efficiency, Data compression formats, GZIP, ZIP, Digital multimedia, Error correction codes.*

1. Xofman kodlashining asosiy prinsiplari

Xofman kodlash tizimi 1952-yilda David A. Huffman tomonidan ishlab chiqilgan. Xofman kodlashining asosiy maqsadi, ma'lumotlarni siqishda har bir belgining kodlarini ehtimollikka mos ravishda taqsimlashdir. Ushbu usulning asosiy tamoyillari quyidagilardan iborat:

- **Ehtimollik asosida kodlash**: Belgilarni kodlashda, eng tez-tez uchraydigan belgilar eng qisqa kodlarga ega bo'ladi, kam uchraydiganlar esa uzunroq kodlarga ega bo'ladi.
- **Prefix kodlash**: Xofman kodlashida, hech qanday kod boshqa bir kodning prefiksi bo'lmaydi. Bu shuni anglatadiki, kodlar orasida o'zaro to'qnashuvlar bo'lmaydi, va har bir kodli ketma-ketlik aniq dekodlanishi mumkin.
- **Kod daraxti**: Xofman algoritmi kodlashning optimal tuzilmasini yaratish uchun ikkita asosiy parametrga asoslanadi: belgilar va ularning paydo bo'lish ehtimollari. Kod daraxti (Huffman tree) bu parametrlar asosida quriladi.

2. Xofman kodlash algoritmi

Xofman kodlash algoritmi quyidagi bosqichlarni o'z ichiga oladi:

1. **Ehtimollarni hisoblash**: Ma'lumotlar to'plamidagi har bir belgining paydo bo'lish ehtimolini aniqlash.
2. **Kod daraxtini qurish**:
    - Har bir belgi va uning ehtimolligini ajratilgan tugun sifatida ko'rib chiqish.

- Eng kam ehtimollikdagi ikki tugunni tanlab, yangi tugun yaratish. Yangi tugun, ushbu ikki tugunning ehtimollarining yig'indisi bo'lib, ularning o'rnida paydo bo'ladi.
- Yangi tugunlar qadam-baqadam qo'shilgan holda daraxtning ildiziga yetguncha jarayon davom etadi.

3. **Kod yaratish**: Kod daraxti qurilganidan so'ng, har bir tugundan 0 yoki 1 ni ajratib, har bir belgi uchun yagona kodni yaratish. Kodlar daraxtning ildizidan har bir bargiga borish yo'li sifatida tashkil etiladi.
4. **Dekodlash**: Xofman kodlashining yana bir muhim jihati – dekodlash jarayoni. Har bir kodning o'zi o'ziga xos bo'lganligi sababli, ularni noto'g'ri tushunish ehtimoli yo'q.

Xofman kodlash tizimi, ma'lumotlarni siqish uchun ishlatiladigan samarali algoritm bo'lib, har bir belgi uchun qisqa kodlar yaratishga asoslanadi. Bu kodlar, belgilarni tasodifiy ravishda bir-biridan farqlash uchun ularning paydo bo'lish ehtimollariga asoslanadi. Keling, Xofman kodlashining qanday ishlashini misol orqali tushuntirib beraman.

Misol: "ABRACADABRA" so'zi uchun Xofman kodlash

1. Belgilar va ularning ehtimolliklarini hisoblash

Avvalo, biz "ABRACADABRA" so'zidagi har bir belgining nechalik tez-tez uchrayotganini hisoblashimiz kerak.

So'zdagi har bir belgi:

- **A**: 5 ta
- **B**: 2 ta
- **R**: 2 ta
- **C**: 1 ta
- **D**: 1 ta

Endi ehtimollarni hisoblaymiz. So'zda jami 11 ta belgi bor, shuning uchun har bir belgining ehtimoli quyidagicha bo'ladi:

- Ehtimol(A) = 5/11
- Ehtimol(B) = 2/11
- Ehtimol(R) = 2/11
- Ehtimol(C) = 1/11
- Ehtimol(D) = 1/11

2. Xofman daraxtini qurish

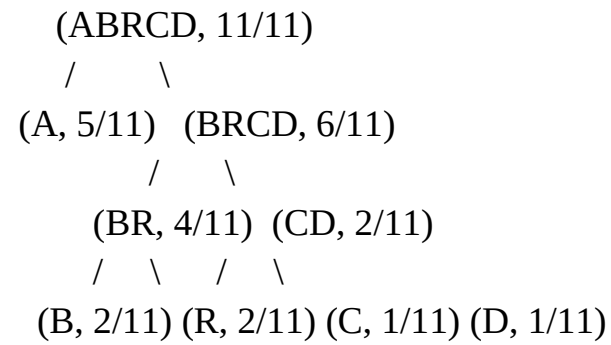
Xofman kodlash algoritmi kod daraxtini quyidagi qadamlar bilan quradi:

1. Har bir belgi va uning ehtimoli bo'yicha tugunlar yaratiladi:
2. (A, 5/11), (B, 2/11), (R, 2/11), (C, 1/11), (D, 1/11)
3. Eng kichik ehtimollikdagi ikkita tugunni tanlab, ulardan yangi bir tugun hosil qilinadi. Bu yangi tugunning ehtimoli, ikkala tugunning ehtimollarining yig'indisi bo'ladi.

Birinchi qadamda C va D tugunlari tanlanadi, ularning ehtimolligi 1/11. Yangi tugun (CD, 2/11) hosil bo'ladi:
(A, 5/11), (B, 2/11), (R, 2/11), (CD, 2/11)

4. Yana ikkita eng kichik ehtimollikdagi tugunlar tanlanadi: B va R (har biri 2/11). Yangi tugun (BR, 4/11) hosil bo'ladi:
5. (A, 5/11), (BR, 4/11), (CD, 2/11)
6. Keyingi qadamda CD (2/11) va (BR, 4/11) tugunlari tanlanadi, ularning ehtimoli 6/11. Yangi tugun (BRCD, 6/11) hosil bo'ladi:
7. (A, 5/11), (BRCD, 6/11)
8. Nihoyat, qolgan ikkita tugun (A, 5/11) va (BRCD, 6/11) tanlanadi va yakuniy tugun (ABRCD, 11/11) hosil bo'ladi.

Endi daraxtimiz quyidagicha bo'ladi:



3. Kodlarni yaratish

Endi kodlarni yaratamiz. Daraxtning har bir shoxchasida 0 yoki 1 belgilangan. Har bir belgi uchun yo'lni aniqlash va kodlarni yozish:

- **A**: 0
- **B**: 10
- **R**: 11
- **C**: 010
- **D**: 011

4. Xofman kodlashini qo'llash

Endi, bizning dastlabki so'zimni (ya'ni "ABRACADABRA") Xofman kodlash yordamida kodlasha olamiz. So'zdagi har bir harf uchun yuqoridagi kodlarni almashtiramiz:

- **A** → 0
- **B** → 10
- **R** → 11
- **A** → 0
- **C** → 010
- **A** → 0
- **D** → 011
- **A** → 0
- **B** → 10
- **R** → 11
- **A** → 0

Natijada quyidagi Xofman kodlangan so'zni olamiz:

0 10 11 0 010 0 011 0 10 11 0

Bu so'zni kodlash orqali biz ma'lumotni siqishga erishdik. Xofman kodlashining afzalligi shundaki, kodlar optimal tarzda taqsimlanadi va eng tez-tez uchraydigan belgilar qisqa kodlarga, kamroq uchraydiganlar esa uzoqroq kodlarga ega bo'ladi.

C# tilida **Xofman kodlash** algoritmini amalga oshirish uchun quyidagi dastur kodini keltiraman. Ushbu dastur, yuqoridagi misolni asos qilib olgan holda, so'zdagi harflarni Xofman algoritmi yordamida kodlash va dekodlashni amalga oshiradi.

C# Dastur Kodeksida Xofman Kodlash

```
using System;
using System.Collections.Generic;
using System.Linq;
class HuffmanNode
{
   public char Character { get; set; }
   public int Frequency { get; set; }
   public HuffmanNode Left { get; set; }
```

```
    public HuffmanNode Right { get; set; }
    public HuffmanNode(char character, int frequency)
    {
        Character = character;
        Frequency = frequency;
    }
    // Constructor for internal nodes
    public HuffmanNode(int frequency, HuffmanNode left, HuffmanNode right)
    {
        Frequency = frequency;
        Left = left;
        Right = right;
    }
}
class HuffmanCoding
{
    // Step 1: Generate the Huffman Tree
    public static HuffmanNode BuildHuffmanTree(Dictionary<char, int> frequencies)
    {
        var priorityQueue = new SortedList<int, List<HuffmanNode>>();
        // Initialize priority queue with single-node trees
        foreach (var kvp in frequencies)
        {
            var node = new HuffmanNode(kvp.Key, kvp.Value);
            if (!priorityQueue.ContainsKey(kvp.Value))
            {
                priorityQueue[kvp.Value] = new List<HuffmanNode>();
            }
            priorityQueue[kvp.Value].Add(node);
        }
        // Step 2: Build the tree by combining the least frequent nodes
        while (priorityQueue.Count > 1)
        {
            var first = priorityQueue.Values[0][0]; // Get first node (smallest frequency)
            var second = priorityQueue.Values[1][0]; // Get second node

            priorityQueue.Values[0].RemoveAt(0);
```

```
            priorityQueue.Values[1].RemoveAt(0);
            int mergedFrequency = first.Frequency + second.Frequency;
            var mergedNode = new HuffmanNode(mergedFrequency, first, second);
            if (!priorityQueue.ContainsKey(mergedFrequency))
            {
                priorityQueue[mergedFrequency] = new List<HuffmanNode>();
            }
            priorityQueue[mergedFrequency].Add(mergedNode);
        }
        return priorityQueue.Values[0][0]; // Return the root of the Huffman Tree
    }
    // Step 3: Generate the Huffman Codes
    public static Dictionary<char, string> GenerateCodes(HuffmanNode root)
    {
        var codes = new Dictionary<char, string>();
        GenerateCodesHelper(root, "", codes);
        return codes;
    }
    private static void GenerateCodesHelper(HuffmanNode node, string code,
Dictionary<char, string> codes)
    {
        if (node == null)
        {
            return;
        }
        if (node.Left == null && node.Right == null) // It's a leaf node
        {
            codes[node.Character] = code;
        }
        GenerateCodesHelper(node.Left, code + "0", codes); // Left child is '0'
        GenerateCodesHelper(node.Right, code + "1", codes); // Right child is '1'
    }
    // Step 4: Encode a string using the Huffman Codes
    public static string Encode(string input, Dictionary<char, string> codes)
    {
        string encodedString = "";
        foreach (char character in input)
        {
            encodedString += codes[character];
```

```
        }
        return encodedString;
    }
    // Step 5: Decode the encoded string
    public static string Decode(string encodedString, HuffmanNode root)
    {
        string decodedString = "";
        HuffmanNode currentNode = root;

        foreach (char bit in encodedString)
        {
            currentNode = bit == '0' ? currentNode.Left : currentNode.Right;
            if (currentNode.Left == null && currentNode.Right == null) // Leaf node
            {
                decodedString += currentNode.Character;
                currentNode = root; // Go back to the root for the next character
            }
        }
        return decodedString;
    }
}
class Program
{
    static void Main(string[] args)
    {
        string input = "ABRACADABRA";
        // Step 1: Count frequencies of each character
        var frequencies = new Dictionary<char, int>();
        foreach (char c in input)
        {
            if (frequencies.ContainsKey(c))
                frequencies[c]++;
            else
                frequencies[c] = 1;
        }
        // Step 2: Build the Huffman Tree
        HuffmanNode root = HuffmanCoding.BuildHuffmanTree(frequencies);
        // Step 3: Generate Huffman Codes
        var codes = HuffmanCoding.GenerateCodes(root);
```

```
        // Step 4: Encode the input string
        string encodedString = HuffmanCoding.Encode(input, codes);
        Console.WriteLine("Encoded String: " + encodedString);
        // Step 5: Decode the encoded string
        string decodedString = HuffmanCoding.Decode(encodedString, root);
        Console.WriteLine("Decoded String: " + decodedString);
    }
}
```

Adabiyotlar

1. Huffman, D. A. (1952). "A Method for the Construction of Minimum-Redundancy Codes." *Proceedings of the IRE*, 40(9), 1098–1101.
2. Salomon, D. (2007). *Data Compression: The Complete Reference*. Springer.
3. Sayood, K. (2017). *Introduction to Data Compression*. Morgan Kaufmann