УДК 004

Реализация алгоритма Хаффмана на языке программирования C++

Романов Даниил Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Целью данной статьи является, создание программы способной сжимать текстовые данные с помощью алгоритма Хаффмана. Программа будет написана на языке программирования C++ в среде разработки Visual Studio. Результатом исследования станет готовая программа с подробным описанием её реализации.
Ключевые слова: сжатие данных, алгоритм Хаффмана, C++, Visual Studio

Implementation of the Huffman algorithm in the C++ programming language

Romanov Daniil Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The purpose of this article is to create a program capable of compressing text data using the Huffman algorithm. The program will be written in the C++ programming language in the Visual Studio development environment. The result of the study will be a ready-made program with a detailed description of its implementation.
Keywords: data compression, Huffman algorithm, C++, Visual Studio

1 Введение

1.1 Актуальность

В современном мире количество информации увеличивается с каждым днём и всё более возникает потребность в сжатии и более эргономичном распределении информации. Для этого существуют разные способы сжатия и оптимизации информации, одним из таких методов является алгоритм Хаффмана. Основная идея алгоритма заключается в том, что наиболее часто встречаемый символ кодируется наименьшим значением, а символ, который встречается реже всего, получает, наоборот, очень длинный код. Таким образом при обработке ввода, наиболее частый символ занимают меньше всего место, что и способствует сжатию данных.

1.2 Обзор исследований

В своей работе О.О. Евсютин описывает различные модели сжатия данных и их применеие [1]. Р.С. Добычин и А.Р. Петров сравнивают

алгоритмы Хаффмана и Шеннона-Фано и объясняют преимущества и недостатки каждого из них [2]. В.Г. Тронин и В.А. Болкунов рассказывают о комбинировании различных методов сжатия данных [3].

1.3 Цель исследования

Цель исследования – создание программы, способной с помощью алгоритма Хаффмана сжимать текстовые данные.

2 Материалы и методы

Для создания программы потребуется среда программирования Visual Studio [4].

3 Результаты и обсуждение

Разработка программы начинается с создания проекта в Visual Studio и импортирования нужных библиотек, таких как iostream, string, queue и unordered_map (рис.1).

```
#include <iostream>
#include <string>
#include <queue>
#include <unordered_map>
#define EMPTY_STRING ""
using namespace std;
```

Рисунок 1 – Импортирование нужных библиотек

Затем создаём узел бинарного дерева (рис.2).

```
struct Node
{
    char ch;
    int freq;
    Node* left, * right;
};
```

Рисунок 2 – Узел бинарного дерева

Создаём функцию для выделения основного узла дерева (рис.3).

```
Node* getNode(char ch, int freq, Node* left, Node* right)
{
    Node* node = new Node();

    node->ch = ch;
    node->freq = freq;
    node->left = left;
    node->right = right;

    return node;
}
```

Рисунок 3 - Функция нового узла

Далее создаём объект сравнения, который будет использоваться для упорядочивания списка. Внутри этого объекта, элемент, который имеет наивысший приоритет будет обладать наименьшей частотой (рис.4).

```
struct comp
{
    bool operator()(const Node* l, const Node* r) const
    {
        return l->freq > r->freq;
    }
};
```

Рисунок 4 – Объект сравнения

Дерево Хаффмана может содержать лишь один узел. Чтобы определить это, следует создать дополнительную функцию. (рис.5).

```
bool isLeaf(Node* root) {
    return root->left == nullptr && root->right == nullptr;
}
```

Рисунок 5 – Вспомогательная функция

Нужно просмотреть дерево Хаффмана и сохранить коды Хаффмана на карте. Если найден листовой узел, то кодируем его как 1 (рис.6).

```
void encode(Node* root, string str, unordered_map<char, string>& huffmanCode)
{
    if (root == nullptr) {
        return;
    }
    if (isLeaf(root)) {
        huffmanCode[root->ch] = (str != EMPTY_STRING) ? str : "1";
    }

    encode(root->left, str + "0", huffmanCode);
    encode(root->right, str + "1", huffmanCode);
}
```

Рисунок 6 – Функция определения узла

Создаём функцию для просмотра по дерева Хаффмана и декодируем закодированную строку (рис.7).

```
void decode(Node* root, int& index, string str)
{
    if (root == nullptr) {
        return;
    }
    if (isLeaf(root))
    {
        cout << root->ch;
        return;
    }
    index++;
    if (str[index] == '0') {
        decode(root->left, index, str);
    }
    else {
        decode(root->right, index, str);
    }
}
```

Рисунок 7 – Функция де кодировки

Далее строим функцию, которая построит дерево Хаффмана и декодирует набранный текст, например, “Carl stole an instrument from Carl, and Clara stole a clarinet from Carl.”. Для начала обрабатывается случай исключение с пустой строкой. Затем рассчитывается частота генерации каждого символа и сохраняется в памяти. Создаем приоритетную очередь для записи активных узлов дерева Хаффмана. Для каждого символа создаем конечный узел и добавляем его в приоритетную очередь, это стоит повторять до того момента, пока в queue не появится более одного узла. Набранные данные могут иметь опечатку по типу: c, cc, ccc, которую тоже стоит обработать (рис.8).

```
void buildHuffmanTree(string text)
{
    if (text == EMPTY_STRING) {
        return;
    }
    unordered_map<char, int> freq;
    for (char ch : text) {
        freq[ch]++;
    }
    priority_queue<Node*, vector<Node*>, comp> pq;
    for (auto pair : freq) {
        pq.push(getNode(pair.first, pair.second, nullptr, nullptr));
    }
    while (pq.size() != 1)
    {
        Node* left = pq.top(); pq.pop();
        Node* right = pq.top();    pq.pop();
        int sum = left->freq + right->freq;
        pq.push(getNode('\0', sum, left, right));
    }
    Node* root = pq.top();
    unordered_map<char, string> huffmanCode;
    encode(root, EMPTY_STRING, huffmanCode);
    cout << "Huffman Codes are:\n" << endl;
    for (auto pair : huffmanCode) {
        cout << pair.first << " " << pair.second << endl;
    }
    cout << "\nThe original string is:\n" << text << endl;
    string str;
    for (char ch : text) {
        str += huffmanCode[ch];
    }
    cout << "\nThe encoded string is:\n" << str << endl;
    cout << "\nThe decoded string is:\n";
    if (isLeaf(root))
    {
        while (root->freq--) {
            cout << root->ch;
        }
    }
    else {
        int index = -1;
        while (index < (int)str.size() - 1) {
            decode(root, index, str);
        }
    }
}
```

Рисунок 8 – Функция алгоритма Хаффмана

Переходим к проверке алгоритма (рис.9).

```
int main()
{
    string text = "Carl stole an instrument from Carl, and Clara stole a clarinet from Carl.";
    buildHuffmanTree(text);

    return 0;
}
```

Рисунок 9 – Проверка алгоритма

Запустив программу, получаем следующие результаты. Размер введённой строки составляет 592(8 * 74) бита, а закодированная строка занимает 278 бита. Наиболее часто встречаемый символ кодируется наименьшим значением, что и выполняется в алгоритме. Тем самым сжатие составило 47% (рис.10).

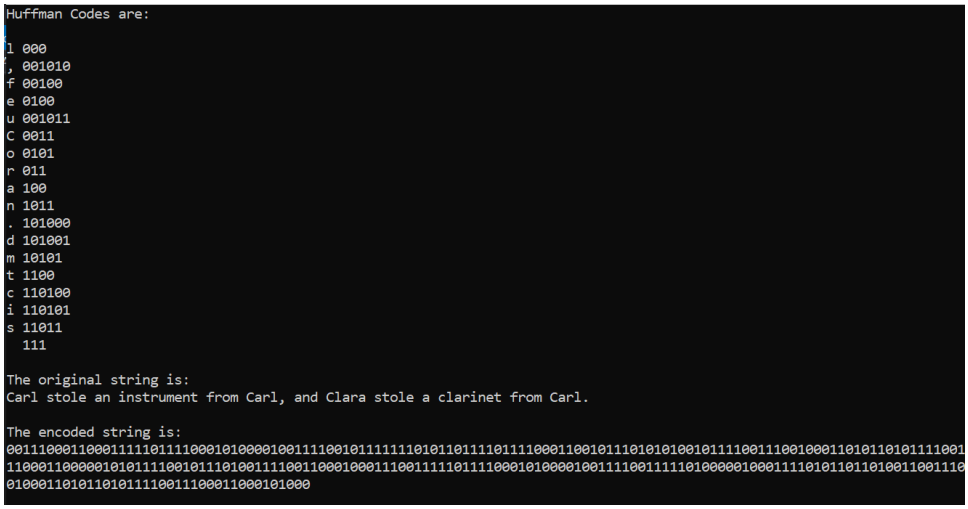

Рисунок 10 – Результат программы

Выводы

В данной работе была создана программа способная сжимать текстовую информацию примерно в 2 раза. Полученный алгоритм может быть усовершенствован и использован для создания более функциональных и удобных приложений по сжатию данных.

Библиографический список

1. Евсютин О. О. Модели сжатия данных //Электронные средства и системы управления. Материалы докладов Международной научно-практической конференции. – федеральное государственное бюджетное образовательное учреждение высшего образования Томский государственный университет систем управления и радиоэлектроники, 2011. №. 1. С. 125-132.
2. Добычин Р. С., Петров А. Р. Сравнительный анализ работы алгоритмов Шеннона-Фано и Хаффмана //Междисциплинарные исследования в области математического моделирования и информатики. 2016. С. 12-14.
3. Тронин В. Г., Болкунов В. А. Комбинирование алгоритмов сжатия данных //Информатика, моделирование, автоматизация проектирования. 2019. С. 169-175.
4. Visual Studio URL: https://visualstudio.microsoft.com/ru