МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №8

Тема: ОТДЕЛЬНЫЕ ВОПРОСЫ АЛГОРИТМИЗАЦИИ

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент: Павлов Никита
группа ИКБО-50-23

Москва 2024

СОДЕРЖАНИЕ

1. АЛГОРИТМЫ КОДИРОВАНИЯ И СЖАТИЯ ДАННЫХ

1.1 ЦЕЛЬ РАБОТЫ

Освоить приёмы работы с алгоритмами кодирования и сжатия.

Персональный вариант: 24

Фраза для кодировки методом Шеннона-Фано: Кони-кони, коникони, Мы сидели на балконе, Чай пили, воду пили, По-турецки говорили.

Фраза для кодировки методам Хаффмана: Павлов Никита Сергеевич

Фраза для кодировки методам LZ77: 0100001000000100001

Фраза для кодировки методам LZ78: пропронепронепрнепрона с.

1.2 КОДИРОВКА ШЕННОНА-ФАНО

1.2.1 Формулировка задачи

Разработать алгоритм и реализовать программу сжатия текста алгоритмом Шеннона – Фано. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на тексте, определенным вариантом. Определить процент сжатия.

1.2.2 Математическая модель решения (описание алгоритма)

Метод Шеннона-Фано — это алгоритм построения префиксных кодов для символов, который используется для сжатия данных. Он работает по следующему принципу:

1. **Подсчет частот:** Сначала подсчитывается частота появления каждого символа в исходном тексте.
2. **Сортировка:** Символы упорядочиваются по убыванию частоты.
3. **Разделение:** Символы разделяются на две группы с максимально близкими суммами частот.
4. **Присвоение кодов:** Символам в первой группе присваивается префиксный код "0", а символам во второй группе - префиксный код "1".
5. **Рекурсивный шаг:** Процесс повторяется рекурсивно для каждой группы, пока не останется один символ.

Пример оформления таблицы кодирования фразы «Кони-кони, коникони, Мы сидели на балконе, Чай пили, воду пили, По-турецки говорили.», используя метод Шеннона–Фано:

Таблица 1 – Таблица кодировки Шеннона-Фано

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	Код	Кол-во бит
К	1	1	0	1	1	1	0	101110	6
М	1	1	1	0	1	0		11010	5
П	1	0	1	0	1	1	0	010110	6
Ч	1	1	0	1	1	1	1	101111	6

а	3	1	0	1	0	0		10100	15
б	1	0	1	0	1	1	1	010111	6
в	2	1	1	1	1	0	0	111100	12
г	1	1	1	1	1	0	1	111101	6
д	2	0	0	1	1	0		00110	10
е	3	1	0	1	1	0		10110	15
и	13	0	0	0				000	39
й	1	0	0	1	0	1	0	001010	6
к	5	0	1	1	0			0110	20
л	5	1	1	1	0			1110	20
н	6	0	1	0	0			0100	24
о	9	1	1	0	0			1100	36
п	2	1	0	1	0	1	0	101010	12
р	2	0	1	0	1	0	0	010100	12
с	1	1	0	0	1			1001	4
т	1	0	1	0	1	0	1	010101	6
у	2	1	1	1	1	1		11111	10
ц	1	1	1	0	1	1		11011	5
ы	1	0	0	1	0	1	1	001011	6
enter	3	0	0	1	0	0		00100	15
пробел	10	1	0	0	0			1000	40
,	5	0	1	1	1			0111	20
-	2	0	0	1	1	1		00111	10
.	1	1	0	1	0	1	1	101011	6
									378

Незакодированная фраза – 84 * 8 бит = 672 бит.

Закодированная фраза – 378 бит.

Коэффициент сжатия – 378 / 672 = 56.25%

1.2.3 Код программы

```
struct Node {
    char symbol;
    int frequency;
    Node* left;
    Node* right;

    Node(char s, int f) : symbol(s), frequency(f), left(nullptr), right(nullptr) {}
};

bool compareNodesByFrequency(Node* a, Node* b) {
    return a->frequency > b->frequency;
}
```

Рисунок 1 – Код программы

```cpp
int calculateTotalFrequency(const vector<Node*>& nodes) {
    int totalFrequency = 0;
    for (Node* node : nodes) {
        totalFrequency += node->frequency;
    }
    return totalFrequency;
}

map<char, int> convertNodesToMap(const vector<Node*>& nodes) {
    map<char, int> frequencies;
    for (Node* node : nodes) {
        frequencies[node->symbol] = node->frequency;
    }
    return frequencies;
}
```

```cpp
Node* createShannonFanoTree(const map<char, int>& frequencies) {
    vector<Node*> nodes;
    for (const auto& pair : frequencies) {
        nodes.push_back(new Node(pair.first, pair.second));
    }
    sort(nodes.begin(), nodes.end(), compareNodesByFrequency);

    while (nodes.size() > 1) {
        vector<Node*> leftSubtree, rightSubtree;
        int summLeft = 0, summRight = 0;
        for (int i = 0; i < nodes.size(); i++) {
            if (summLeft <= summRight) {
                leftSubtree.push_back(nodes[i]);
                summLeft += nodes[i]->frequency;
            }
            else {
                rightSubtree.push_back(nodes[i]);
                summRight += nodes[i]->frequency;
            }
        }

        Node* parent = new Node('\0', calculateTotalFrequency(leftSubtree) + calculateTotalFrequency(rightSubtree));
        parent->left = createShannonFanoTree(convertNodesToMap(leftSubtree));
        parent->right = createShannonFanoTree(convertNodesToMap(rightSubtree));

        nodes.clear();
        nodes.push_back(parent);
        sort(nodes.begin(), nodes.end(), compareNodesByFrequency);
    }

    return nodes.front();
}
```

```cpp
void traverse(Node* node, string code, map<char, string>& codes) {
    if (node->left == nullptr && node->right == nullptr) {
        codes[node->symbol] = code;
        return;
    }
    traverse(node->left, code + "0", codes);
    traverse(node->right, code + "1", codes);
};

map<char, string> getCodes(Node* root) {
    map<char, string> codes;
    if (root == nullptr) {
        return codes;
    }
    traverse(root, "", codes);
    return codes;
}
```

Рисунок 2-4 – Код программы

```
string compress(const string& text, const map<char, string>& codes) {
    string compressedText;
    for (char c : text) {
        compressedText += codes.at(c);
    }
    return compressedText;
}

string decompress(const string& compressedText, const map<char, string>& codes) {
    string decompressedText = "";
    int i = 0;
    string currentCode = "";
    while (i < compressedText.length()) {
        currentCode += compressedText[i];
        for (const auto& pair : codes) {
            if (pair.second == currentCode) {
                decompressedText += pair.first;
                currentCode = "";
                break;
            }
        }
        i++;
    }
    return decompressedText;
}
```

```
void shannonFano() {
    cout << "Реализация сжатия методом Шеннона-Фано:\n\n";

    string text = "Кони-кони, коникони,\nМы сидели на балконе,\nЧай пили, воду пили,\nПо - турецки говорили.";
    cout << "Сжимаемые данные:" << text << endl;

    map<char, int> frequency;
    for (char c : text) {
        frequency[c]++;
    }

    Node* root = createShannonFanoTree(frequency);

    map<char, string> codes = getCodes(root);

    /*cout << "Таблица кодировки символов:" << endl;
    for (auto code : codes) {
        cout << code.first << " " << code.second << endl;
    }*/

    string compressedText = compress(text, codes);
    cout << "Сжатый текст: " << compressedText << endl;

    string decompressedText = decompress(compressedText, codes);
    cout << "Восстановленный текст: " << decompressedText << endl << endl << endl;
}
```

Рисунок 5-6 – Код программы

1.2.4 Результаты тестирования

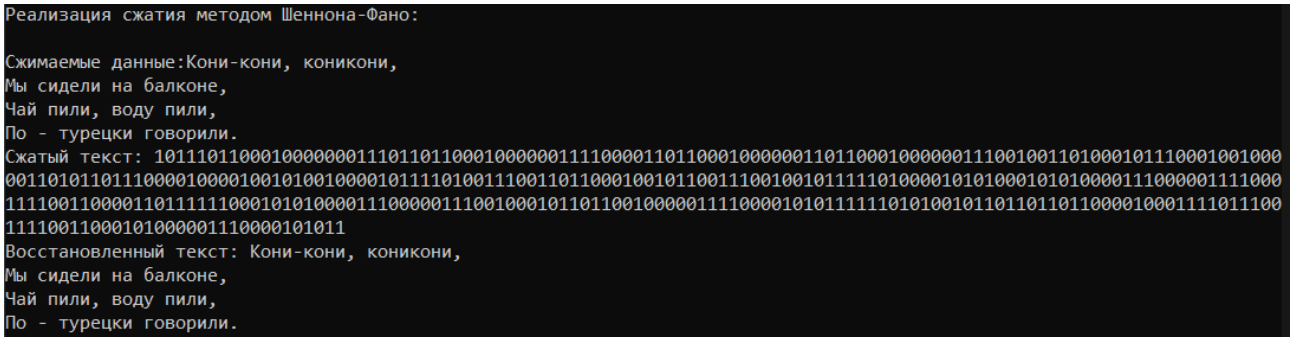

Рисунок 7 – Результаты тестирования

1.3 КОДИРОВКА ХАФФМАНА

1.2.1 Формулировка задачи

Разработать алгоритм и реализовать программу сжатия текста алгоритмом Хаффмана. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на тексте, определенным вариантом. Определить процент сжатия.

1.2.2 Математическая модель решения (описание алгоритма)

Метод Хаффмана — это алгоритм построения префиксных кодов для символов, который используется для сжатия данных. Он работает по следующему принципу:

1. **Подсчет частот:** Сначала подсчитывается частота появления каждого символа в исходном тексте.
2. **Сортировка:** Символы упорядочиваются по убыванию частоты.
3. **Объединение:** Два узла с наименьшей частотой объединяются в новый родительский узел. Частота родительского узла равна сумме частот исходных узлов.
4. **Повторение операций:** Процесс сортировки и объединения повторяется, пока не останется один символ.
5. **Присвоение кодов:** Рекурсивно запускается процесс присвоения кодов. Символам в левой ветке присваивается префиксный код "0", а символам в правой ветке - префиксный код "1".

Пример оформления графа кодирования фразы «Павлов Никита Сергеевич», используя метод Хаффмана:

Незакодированная фраза – 23 * 8 бит = 184 бит.

Закодированная фраза – 87 бит.

Коэффициент сжатия – 87 / 184 = 47.28%

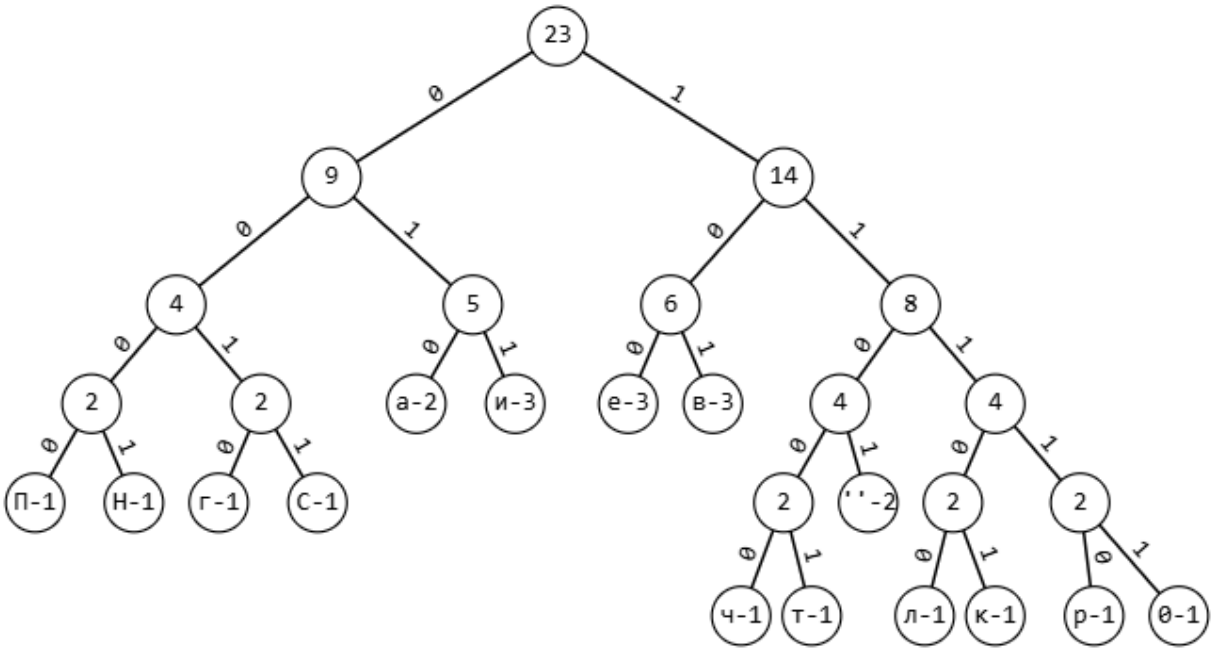

Рисунок 8 – Дерево Хаффмана

1.2.3 Код программы

```
Node* createHuffmanTree(const map<char, int>& frequencies) {
    vector<Node*> nodes;
    for (const auto& pair : frequencies) {
        nodes.push_back(new Node(pair.first, pair.second));
    }
    sort(nodes.begin(), nodes.end(), compareNodesByFrequency);

    while (nodes.size() > 1) {
        Node* left = nodes.back();
        nodes.pop_back();
        Node* right = nodes.back();
        nodes.pop_back();
        Node* parent = new Node('\0', left->frequency + right->frequency);
        parent->left = left;
        parent->right = right;
        nodes.push_back(parent);
        sort(nodes.begin(), nodes.end(), compareNodesByFrequency);
    }

    return nodes.front();
}
```

Рисунок 9 – Код программы

```
void huffman() {
    cout << "Реализация сжатия методом Хаффмана:\n\n";

    string text = "Павлов Никита Сергеевич";
    cout << "Сжимаемые данные:" << text << endl;

    map<char, int> frequency;
    for (char c : text) {
        frequency[c]++;
    }

    Node* root = createHuffmanTree(frequency);

    map<char, string> codes = getCodes(root);

    /*cout << "Таблица кодировки символов:" << endl;
    for (auto code : codes) {
        cout << code.first << " " << code.second << endl;
    }*/

    string compressedText = compress(text, codes);
    cout << "Сжатый текст: " << compressedText << endl;

    string decompressedText = decompress(compressedText, codes);
    cout << "Восстановленный текст: " << decompressedText << endl << endl << endl;
}
```

Рисунок 10 – Код программы

1.2.4 Результаты тестирования

```
Реализация сжатия методом Хаффмана:

Сжимаемые данные:Павлов Никита Сергеевич
Сжатый текст: 0000010101111001111110111010001011111010111100101011010011100111100010100100101011110000
Восстановленный текст: Павлов Никита Сергеевич
```

Рисунок 11 – Результаты тестирования

1.4 КОДИРОВКА LZ77

1.2.1 Формулировка задачи

Разработать алгоритм и реализовать программу сжатия текста алгоритмом LZ77. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на тексте, определенным вариантом. Определить процент сжатия.

1.2.2 Математическая модель решения (описание алгоритма)

LZ77 использует скользящее по сообщению окно. Не использует словарь. Допустим, на текущей итерации окно зафиксировано. С правой стороны окна наращиваем подстроку, пока она есть в строке <скользящее окно + наращиваемая строка> и начинается в скользящем окне. Назовем наращиваемую строку буфером. После наращивания алгоритм выдает код <offset,length,nextChar> состоящий из трех элементов:

- смещение в окне - offset;
- длина буфера - length;
- первый не совпавший символ буфера - nextChar.

В конце итерации алгоритм сдвигает окно на length+1.

Рассмотрим процесс кодирования строки «0100001000000100001»:

Таблица 2 – Таблица кодировки LZ77:

Шаг	Скользящее окно		Совпадающая фраза	Закодированные данные		
	Словарь	Буфер		offset	length	nextChar
1	-	01000011	-	0	0	0
2	0	10000100	-	0	0	1
3	01	00001000	0	2	1	0
4	0100	00100000	00	2	2	1
5	0100001	00000010	0000	5	4	0
6	010000100000	0100001-	0100001	12	7	-

Незакодированная фраза – 19 * 8 бит = 152 бит.

Закодированная фраза – 6 * 3 * 8 бит = 144 бит.

Коэффициент сжатия – 144 / 152 = 94.74%

1.2.3 Код программы

```
struct LZ77Entry {
    int offset;
    int length;
    char nextChar;
};

vector<LZ77Entry> compressLZ77(const string& data) {
    vector<LZ77Entry> compressedData;
    int windowSize = 1024;
    int searchBufferSize = 4096;
    int i = 0;

    while (i < data.length()) {
        int maxMatchLength = 0;
        int maxMatchOffset = 0;
        for (int j = max(0, i - windowSize); j < i; j++) {
            int k = 0;
            while (j + k < i && data[j + k] == data[i + k] && k < searchBufferSize) {
                k++;
            }
            if (k >= maxMatchLength) {
                maxMatchLength = k;
                maxMatchOffset = i - j;
            }
        }
        if (maxMatchLength > 0) {
            compressedData.push_back({ maxMatchOffset, maxMatchLength, data[i + maxMatchLength] });
            i += maxMatchLength + 1;
        }
        else {
            compressedData.push_back({ 0, 0, data[i] });
            i++;
        }
    }

    return compressedData;
}
```

Рисунок 12 – Код программы

```cpp
string decompressLZ77(const vector<LZ77Entry>& compressedData) {
    string decompressedData;

    for (const auto& entry : compressedData) {
        if (entry.length > 0) {
            int start = decompressedData.size() - entry.offset;
            for (int i = 0; i < entry.length; i++) {
                decompressedData += decompressedData[start + i];
            }
        }
        decompressedData += entry.nextChar;
    }

    return decompressedData;
}

void lz77() {
    cout << "Реализация сжатия методом LZ77:\n\n";

    string data = "0100001000000100001";
    cout << "Сжимаемые данные: " << data << endl;

    vector<LZ77Entry> compressedData = compressLZ77(data);

    cout << "Сжатые данные:" << endl;
    for (const auto& entry : compressedData) {
        cout << "(" << entry.offset << ", " << entry.length << ", " << entry.nextChar << ")" << endl;
    }

    string decompressedData = decompressLZ77(compressedData);
    cout << "Восстановленные данные: " << decompressedData << endl << endl << endl;
}
```

Рисунок 13 – Код программы

1.2.4 Результаты тестирования

```
Реализация сжатия методом LZ77:

Сжимаемые данные: 0100001000000100001
Сжатые данные:
(0, 0, 0)
(0, 0, 1)
(2, 1, 0)
(2, 2, 1)
(5, 4, 0)
(12, 7, )
Восстановленные данные: 0100001000000100001
```

Рисунок 14 – Результаты тестирования

1.5 КОДИРОВКА LZ78

1.2.1 Формулировка задачи

Разработать алгоритм и реализовать программу сжатия текста алгоритмом LZ78. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на тексте, определенным вариантом. Определить процент сжатия.

1.2.2 Математическая модель решения (описание алгоритма)

В отличие от LZ77, работающего с уже полученными данными, LZ78 ориентируется на данные, которые только будут получены (LZ78 не использует скользящее окно, он хранит словарь из уже просмотренных фраз). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Если в конце алгоритма мы не находим символ, нарушивший совпадения, то тогда мы выдаем код в виде (индекс строки в словаре без последнего символа, последний символ).

Рассмотрим процесс кодирования строки «пропронепронепрнепрона с»:

Таблица 3 – Таблица кодировки LZ78:

Словарь	Считываемое содержимое	Код	
		index	nextChar
	п	0	п
п(1)	р	0	р
п(1), р(2)	о	0	о
п(1), р(2), о(3)	пр	1	р
п(1), р(2), о(3), пр(4)	он	3	н
п(1), р(2), о(3), пр(4), он(5),	е	0	е

п(1), р(2), о(3), пр(4), он(5), е(6)	про	4	о
п(1), р(2), о(3), пр(4), он(5), е(6), про(7)	н	0	н
п(1), р(2), о(3), пр(4), он(5), е(6), про(7), н(8)	еп	6	п
п(1), р(2), о(3), пр(4), он(5), е(6), про(7), н(8), еп(9)	рн	2	н
п(1), р(2), о(3), пр(4), он(5), е(6), про(7), н(8), еп(9), рн(10)	епр	9	р
п(1), р(2), о(3), пр(4), он(5), е(6), про(7), н(8), еп(9), рн(10), епр(11)	она	5	а
п(1), р(2), о(3), пр(4), он(5), е(6), про(7), н(8), еп(9), рн(10), епр(11), она(12)	‘’	0	‘’
п(1), р(2), о(3), пр(4), он(5), е(6), про(7), н(8), еп(9), рн(10), епр(11), она(12), ‘’(13)	с	0	с

Незакодированная фраза – 24 * 8 бит = 192 бит.

Закодированная фраза – 14 * 2 * 8 бит = 224 бит.

Коэффициент сжатия – 224 / 192 = 116.67%

1.2.3 Код программы

```
struct LZ78Entry {
    int index;
    char nextChar;
};

vector<LZ78Entry> compressLZ78(const string& data) {
    vector<LZ78Entry> compressedData;
    unordered_map<string, int> dictionary;
    dictionary[""] = 0;
    int index = 1;
    string currentWord = "";

    for (int i = 0; i < data.length(); i++) {
        string newWord = currentWord + data[i];
        if (dictionary.find(newWord) == dictionary.end()) {
            dictionary[newWord] = index;
            index++;
            compressedData.push_back({ dictionary[currentWord], data[i] });
            currentWord = "";
        }
        else {
            currentWord = newWord;
        }
    }
    if (!currentWord.empty()) {
        compressedData.push_back({ dictionary[currentWord], '\0' });
    }

    return compressedData;
}
```

```
string decompressLZ78(const vector<LZ78Entry>& compressedData) {
    string decompressedData, currentWord;
    unordered_map<int, string> dictionary;
    dictionary[0] = "";

    for (const auto& entry : compressedData) {
        currentWord = dictionary[entry.index] + entry.nextChar;
        decompressedData += currentWord;
        dictionary[dictionary.size()] = currentWord;
    }

    return decompressedData;
}

void lz78() {
    cout << "Реализация сжатия методом LZ78:\n\n";

    string data = "пропронепронепрнепрона с";
    cout << "Сжимаемые данные: " << data << endl;

    vector<LZ78Entry> compressedData = compressLZ78(data);
    cout << "Сжатые данные:" << endl;
    for (const auto& entry : compressedData) {
        cout << "(" << entry.index << ", " << entry.nextChar << ")" << endl;
    }

    string decompressedData = decompressLZ78(compressedData);
    cout << "Восстановленные данные: " << decompressedData << endl << endl << endl;
}
```

Рисунок 15-16 – Код программы

1.2.4 Результаты тестирования

```
Реализация сжатия методом LZ78:

Сжимаемые данные: пропронепронепрнепрона с
Сжатые данные:
(0, п)
(0, р)
(0, о)
(1, р)
(3, н)
(0, е)
(4, о)
(0, н)
(6, п)
(2, н)
(9, р)
(5, а)
(0,  )
(0, с)
Восстановленные данные: пропронепронепрнепрона с
```

Рисунок 17 – Результаты тестирования

1.6 ВЫВОДЫ

Цель работы и сопутствующие задачи были успешно выполнены. Освоены приёмы работы с алгоритмами кодирования и сжатия. Реализованы алгоритмы кодирования методами Шеннона-Фано, Хаффмана, LZ77 и LZ78. Проведена оценка эффективности сжатия каждого метода.

2. РЕАЛИЗАЦИЯ АЛГОРИТМОВ НА ОСНОВЕ СОКРАЩЕНИЯ ЧИСЛА ПЕРЕБОРОВ

2.1 ЦЕЛЬ РАБОТЫ

Освоить приёмы работы с алгоритмами, основанными на сокращении числа переборов.

Персональный вариант: 24

Метод: Динамическое программирование

Задача: Имеется рюкзак с ограниченной вместимостью по массе; также имеется набор вещей с определенным весом и ценностью. Необходимо подобрать такой набор вещей, чтобы он помещался в рюкзаке и имел максимальную ценность (стоимость).

2.2 ЗАДАНИЕ

2.2.1 Постановка задачи

Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.

Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы. Сравнить с числом переборов при применении метода.

2.2.2 Математическая модель решения

1. Инициализация:

- `weights` и `values` - векторы, хранящие веса и ценности предметов соответственно.

- `capacity` - вместимость рюкзака.

- `dp` - двумерный вектор, который будет использоваться для хранения результатов подзадач (таблица динамического программирования).

- `dp[i][w]` - максимальная ценность, которую можно получить, используя первые `i` предметов и имея вместимость рюкзака `w`.

2. Итерация по таблице:

- Внешний цикл `for` (`i = 1; i <= n; i++`) перебирает предметы от 1 до `n`.

- Внутренний цикл `for` (`w = 1; w <= capacity; w++`) перебирает возможные вместимости рюкзака от 1 до `capacity`.

3. Рекурсивное соотношение:

- `if (weights[i - 1] <= w)`: Если текущий предмет `i` помещается в рюкзак с вместимостью `w`, мы должны выбрать максимальную ценность между двумя вариантами:

- Взять текущий предмет (`values[i - 1] + dp[i - 1][w - weights[i - 1]]`) - добавляем ценность текущего предмета к максимальной ценности, которую можно получить, используя предыдущие `i - 1` предметов и вместимость `w - weights[i - 1]`.

- Не брать текущий предмет (`dp[i - 1][w]`) - используем максимальную ценность, полученную с использованием предыдущих `i - 1` предметов и вместимости `w`.

- `else`: Если текущий предмет не помещается в рюкзак, максимальная ценность не меняется и равна `dp[i - 1][w]`.

4. Возврат результата:

- Функция `backpack` возвращает `dp[n][capacity]`, что является максимальной ценностью, которую можно получить, используя все `n` предметов и имея вместимость рюкзака `capacity`.

2.2.3 Код программы

```cpp
#include <iostream>
#include <vector>

using namespace std;

int backpack(const vector<int>& weights, const vector<int>& values, int capacity) {
    int n = weights.size();
    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));
    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= capacity; w++) {
            if (weights[i - 1] <= w) {
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
            }
            else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }
    return dp[n][capacity];
}
```

```cpp
int main() {
    setlocale(LC_ALL, "rus");

    int capacity, n;
    cout << "Введите вместимость рюкзака: ";
    cin >> capacity;
    cout << "Введите количество вещей: ";
    cin >> n;
    vector<int> weights(n), values(n);
    cout << "Введите веса вещей: ";
    for (int i = 0; i < n; i++) {
        cin >> weights[i];
    }
    cout << "Введите ценности вещей: ";
    for (int i = 0; i < n; i++) {
        cin >> values[i];
    }

    int max_value = backpack(weights, values, capacity);
    cout << "Максимальная ценность: " << max_value << endl;

    return 0;
}
```

Рисунок 12-13 – Код программы

2.2.4 Результаты тестирования

```
Введите вместимость рюкзака: 50
Введите количество вещей: 4
Введите веса вещей: 10 20 30 40
Введите ценности вещей: 60 100 120 140
Максимальная ценность: 220
```

Рисунок 16 – Результаты тестирования

2.3 ВЫВОДЫ

Цель работы и сопутствующие задачи были успешно выполнены. Освоены приёмы работы с алгоритмами, основанными на сокращении числа переборов. Успешно решена задача с использованием метода динамического программирования.

СПИСОК ИНФОРМАЦТОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных. Часть 1. Сложность алгоритмов. Сортировки. Линейные структуры данных. Поиск в таблице. : учеб. пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова ; МИРЭА – Российский технологический университет, 2022. – 109 с. – ISBN 978-5-7339-1612-5.

2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01-01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с.