МИНОБРНАУКИ РОССИИ

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: Враженко Д.О.

Группа: ИКБО-50-23

Вариант: 6

Москва – 2024

ЦЕЛЬ РАБОТЫ

Часть 8.1. Освоить алгоритмы кодирования и сжатия данных.

Часть 8.2. Освоить реализацию алгоритмов на основе сокращения числа переборов.

ХОД РАБОТЫ

Часть 8.1. Алгоритмы кодирования и сжатия данных

Формулировка задачи:

Задание 1: Исследование алгоритмов сжатия на примерах.

1) Выполнить каждую задачу варианта, представив алгоритм решения в виде таблицы и указав результат сжатия.

2) Описать процесс восстановления сжатого текста.

Закодировать фразу методами Шеннона-Фано: По-турецки говорили. Чяби, чяряби Чяряби, чяби-чяби. Мы набрали в рот воды.

Ход решения:

В таблице 1 приведена кодировка символов по методу Шеннона – Фано.

Таблица 1 - Кодировка по методу Шеннона – Фано

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	7-я цифра	Код	Кол-во бит
пробел	10	0	0	0					000	30
и	9	0	0	1					001	27
я	7	0	1	0					010	21
б	6	0	1	1	0				0110	24
р	6	0	1	1	1				0111	24
о	5	1	0	0	0				1000	20
.	3	1	0	0	1				1001	12
в	3	1	0	1	0	0			10100	15
ч	3	1	0	1	0	1			10101	15
,	2	1	0	1	1	0			10110	10
-	2	1	0	1	1	1			10111	10
Ч	2	1	1	0	0	0	0		110000	12

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	7-я цифра	Код	Кол-во бит
а	2	1	1	0	0	0	1		110001	12
л	2	1	1	0	0	1			11001	10
т	2	1	1	0	1	0			11010	10
ы	2	1	1	0	1	1			11011	10
М	1	1	1	1	0	0	0	0	1110000	7
П	1	1	1	1	0	0	0	1	1110001	7
г	1	1	1	1	0	0	1		111001	6
д	1	1	1	1	0	1	0		111010	6
е	1	1	1	1	0	1	1		111011	6
к	1	1	1	1	1	0	0		111100	6
н	1	1	1	1	1	0	1		111101	6
у	1	1	1	1	1	1	0		111110	6
ц	1	1	1	1	1	1	1		111111	6

Незакодированная фраза: (10 * 1 + 9 * 1 + 7 * 1 + 6 * 2 + 5 * 1 + 3 * 3 + 2 * * 7 + 1 * 9) * 8 бит = 75 * 8 бит = 600 бит.

Закодированная фраза: (30 + 27 + 21 + 24 + 24 + 20 + 12 + 15 + 15 + 10 + + 10 + 12 + 12 + 10 + 10 + 10 + 7 + 7 + 6 + 6 + 6 + 6 + 6 + 6 + 6) бит = 318 бит.

Для восстановления текста необходимо посимвольно сравнивать закодированную строку с кодами до нахождения совпадения. В случае совпадения обнулять буфер сравнения, а найденное совпадение сохранять.

Сжатие данных по методу Лемпеля–Зива LZ77: Используя двухсимвольный алфавит (0, 1) закодировать следующую фразу: 000101110110100111.

Ход решения:

В таблице 2 приведено сжатие данных по методу Лемпеля – Зива LZ77.

Таблица 2 - Сжатие данных по методу Лемпеля – Зива LZ77

Словарь	Буфер	Совпадение	Код
-	00010100	-	<0, 0, “0”>
0	00101001	0	<1, 1, “0”>
000	10100101	-	<0, 0, “1”>
0001	01001010	01	<2, 2, “0”>
0001010	01010100	01010	<5, 5, “1”>
0001010010101	001101	001	<7, 3, “1”>
00010100101010011	01	01	<3, 2, “-”>

Результат:

<0, 0, “0”><1, 1, “0”><0, 0, “0”><2, 2, “0”><5, 5, “1”><7, 3, “1”><3, 2, “-”>

Восстановление текста:

<0, 0, “0”> - 0

<1, 1, “0”> - 000

<0, 0, “1”> - 0001

<2, 2, “0”> - 0001010

<5, 5, “1”> - 0001010010101

<7, 3, “1”> - 00010100101010011

<3, 2, “-”> - 0001010010101001101

Закодировать следующую фразу, используя код LZ78: менменаменаме-натеп.

Ход решения:

В таблице 3 приведена кодировка фразы с использованием LZ78.

Таблица 3 - Сжатие данных по методу Лемпеля – Зива LZ77

Словарь	Считываемое	Код
	м	<0, “м”>
1: м;	е	<0, “е”>

Словарь	Считываемое	Код
1: м; 2: е;	н	<0, “н”>
1: м; 2: е; 3: н;	ме	<1, “е”>
1: м; 2: е; 3: н; 4: ме;	на	<3, “а”>
1: м; 2: е; 3: н; 4: ме; 5: на;	мен	<4, “н”>
1: м; 2: е; 3: н; 4: ме; 5: на; 6: мен;	а	<0, “а”>
1: м; 2: е; 3: н; 4: ме; 5: на; 6: мен; 7: а;	мена	<6, “а”>
1: м; 2: е; 3: н; 4: ме; 5: на; 6: мен; 7: а;	т	<0, “т”>

Словарь	Считываемое	Код
8: мена;		
1: м; 2: е; 3: н; 4: ме; 5: на; 6: мен; 7: а; 8: мена; 9: т;	еп	<2, “п”>

Результат:

<0, “м”><0, “е”><0, “н”><1, “е”><3, “а”><4, “н”><0, “а”><6, “а”><0, “т”>

<2, “п”>

Восстановление текста:

<0, “м”> - м

<0, “е”> - ме

<0, “н”> - мен

<1, “е”> - менме

<3, “а”> - менмена

<4, “н”> - менменамен

<0, “а”> - менменамена

<6, “а”> - менменаменамена

<0, “т”> - менменаменаменат

<2, “п”> - менменаменаменатеп

Задание 2: Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона – Фано.

Ход решения:

Первым делом формируется структура данных, хранящая коды для каждого встречающегося символа. Согласно алгоритму считается количество вхожде-

ний каждого символа, после чего символы сортируются в порядке невозрастания их количеств вхождений в строку и делятся на две группы с примерно равным суммарным количеством.

После этого алгоритм рекурсивно повторяется для каждой из двух получившихся групп, условием остановки будет являться размер группы, равный единице, иначе же верхней группе добавится в конец 0, нижней – 1.

Для кодирования каждый символ заменяется своим кодом.

Для декодирования набирается буферный код, пока он не совпадёт с таковым в словаре. После найденный символ заменяет свой код, а буферный код сбрасывается в пустую строку.

Код программы с комментариями:

На рис. 1 представлены коды функций для работы со структурой let, в которой хранятся данные о символах, то есть сами символы и их количество. Функция compareLets сравнивает символы между собой и возвращает значение true или false в зависимости от того какой символ «больше» в данном типе сравнения. Функция rec записывает код символа в будущую закодированную строку.

```
struct let
{
    char lt;
    int cnt;
    let(char lt, int cnt)
    {
        this->lt = lt;
        this->cnt = cnt;
    }
};

bool compareLets(const let *pr1, const let *pr2)
{ return pr1->cnt > pr2->cnt; }

void rec(int st, int fn, int sm[], map<char, string> &codes, vector<let *> counts)
{
    if (st >= fn)
        return;
    char r = '0';
    int ed = fn;
    for (int i = st; i <= fn; ++i)
    {
        codes[counts[i]->lt] += r;
        if (i == fn || fn - st == 1 || (sm[i + 1] - sm[st]) > (sm[fn + 1] - sm[i + 1]))
        { r = '1'; ed = i; break; }
    }
    for (int i = ed + 1; i <= fn; ++i)
        codes[counts[i]->lt] += r;
    rec(st, ed, sm, codes, counts);
    rec(ed + 1, fn, sm, codes, counts);
}
```

Рисунок 1 - Структура let и функции compareLets и rec

На рис. 2 представлен код функции makeMap, в которой из строки, передающейся в аргументе функции, создается массив из кодов элементов для каждого символа в исходной строке и этот массив возвращается в конце выполнения кода.

```
map<char, string> makeMap(string text)
{
    map<char, string> codes;
    vector<let *> counts;
    string setText = "";
    for (int i = 0; i < text.length(); ++i)
        if (count(setText.begin(), setText.end(), text[i]) == 0)
        { setText += text[i]; codes[text[i]] = ""; }
    for (int i = 0; i < setText.length(); ++i)
        counts.push_back(new let(setText[i], count(text.begin(), text.end(), setText[i])));
    sort(counts.begin(), counts.end(), compareLets);
    int sm[counts.size() + 1];
    sm[0] = 0;
    for (int i = 0; i < counts.size(); ++i)
        sm[i + 1] = sm[i] + counts[i]->cnt;
    int st = 0;
    int fn = counts.size() - 1;
    char r = '0';
    int ed = 0;
    rec(st, fn, sm, codes, counts);
    return codes;
}
```

Рисунок 2 - Код функции makeMap

На рис. 3 представлены коды функций code и decode, которые кодируют и декодируют текст с помощью массива из кодов элементов, который создаётся в функции makeMap.

```
string code(string text, map<char, string> codes)
{
    string s = "";
    for (int i = 0; i < text.length(); ++i)
        s += codes[text[i]];
    return s;
}

string decode(string text, map<char, string> codes)
{
    string s = "";
    string buffer = "";
    for (int i = 0; i < text.length(); ++i)
    {
        buffer += text[i];
        for (auto &[lt, code] : codes)
            if (code == buffer)
            {
                s += lt;
                buffer = "";
                break;
            }
    }
    return s;
}
```

Рисунок 3 - Коды функций code и decode

На рис. 4 представлены структура haf, которая нужна нам для сжатия и восстановления текста методом Хаффмана, в ней у каждого символа есть связь с соседними символами, а также код. В конструкторе структуры мы вычисляем количество символов и связываем их с соседними, а также присваиваем код, переданный в аргументе. В методе find мы находим символ по его коду.

```
struct haf
{
    char lt;
    char code;
    int cnt;
    haf *left;
    haf *right;
    haf(char lt, int cnt, haf *left, haf *right, char code)
    {
        this->lt = lt;
        if (left && right)
            this->cnt = left->cnt + right->cnt;
        else
            this->cnt = cnt;
        this->left = left;
        this->right = right;
        this->code = code;
    }
    char find(string code, int ind = 0)
    {
        if (!this->left && !this->right)
        {
            if (code.length() == ind)
                return this->lt;
            else
                return '@';
        }
        if (ind >= code.length())
            return '@';
        if (code[ind] == '0')
            return this->left->find(code, ind + 1);
        return this->right->find(code, ind + 1);
    }
```

Рисунок 4 - Структура haf, её конструктор и метода find

На рис. 5 представлены коды метода getCode структуры haf, который возвращает код символа, передающегося в аргументе. Также на рисунке показаны коды функции compareHafs, который сравнивает между собой два символа, и

функции makeHaf, которая возвращает указатель на элемент структуры haf, который хранит в себе данные о нынешнем символе.

```
    string getCode(char lt) {
        if (!this->left && !this->right)
        {
            if (this->lt == lt)
                return string(1, this->code);
            else
                return "";
        }

        string s = this->left->getCode(lt) + this->right->getCode(lt);
        if (s != "" && this->code != '@')
            return this->code + s;
        return s;
    }
};

bool compareHafs(const haf *pr1, const haf *pr2)
{ return pr1->cnt < pr2->cnt; }

haf *makeHaf(string text)
{
    vector<haf *> cnts;
    string setText = "";
    for (int i = 0; i < text.length(); ++i)
        if (count(setText.begin(), setText.end(), text[i]) == 0)
            setText += text[i];

    for (int i = 0; i < setText.length(); ++i)
        cnts.push_back(new haf(setText[i], count(text.begin(), text.end(), setText[i]), nullptr, nullptr, '@'));
    haf *ptr;
    while (cnts.size() > 1)
    {
        sort(cnts.begin(), cnts.end(), compareHafs);
        cnts[0]->code = '0';
        cnts[1]->code = '1';
        ptr = new haf('@', 0, cnts[0], cnts[1], '@');
        cnts.push_back(ptr);
        cnts.erase(cnts.begin());
        cnts.erase(cnts.begin());
    }
    ptr = cnts[0];
    return ptr;
}
```

Рисунок 5 - Коды метода getCode структуры haf и функций compareHaf и makeHaf

На рис. 6 представлены коды функций codeH и decodeH, которые кодируют и декодируют символы в исходной строке.

```
string codeH(string text, haf* codes)
{
    string s = "";
    for (int i = 0; i < text.length(); ++i)
        s += codes->getCode(text[i]);
    return s;
}

string decodeH(string text, haf* codes)
{
    string s = "";
    string buffer = "";
    char lt = '@';
    for (int i = 0; i < text.length(); ++i)
    {
        buffer += text[i];
        lt = codes->find(buffer);
        if (lt != '@')
        {
            s += lt;
            buffer = "";
        }
    }
    return s;
}
```

Рисунок 6 - Коды функций codeH и decodeH

На рис. 7 представлен код main функции.

```
int main()
{
    string s = "По-турецки говорили. Чяби, чяряби Чяряби, чяби-чяби. Мы набрали в рот воды.";
    map<char, string> codes = makeMap(s);
    string codedS = code(s, codes);
    string decodedS = decode(codedS, codes);
    cout << "Кодирование методом Шеннона-Фано:\n Исходная строка:\n Размер в битах: " << s.length() * 8 << endl << s
         << "\n\n Закодированная строка:\n Размер в битах: " << codedS.length() << endl << codedS << "\n\n Раскодированная строка:\n" << decodedS;

    cout << emdl << endl << endl;

    s = "Враженко Даниил Олегович";
    haf *codesH = makeHaf(s);
    codedS = codeH(s, codesH);
    decodedS = decodeH(codedS, codesH);
    cout << "Кодирование методом Хаффмана:\n Исходная строка:\n Размер в битах: " << s.length() * 8 << endl << s
         << "\n\n Закодированная строка:\n Размер в битах: " << codedS.length() << endl << codedS << "Раскодированная строка:\n" << decodedS;
    return 0;
}
```

Рисунок 7 - Код main функции

Результаты тестирования:

На рис. 8 представлен тест программы для метода Шеннона – Фано.

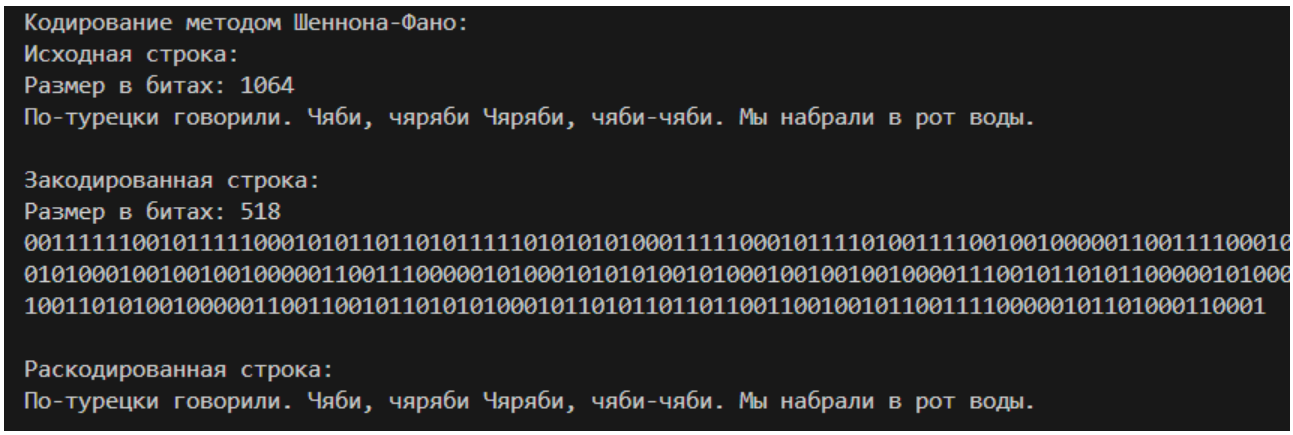

Рисунок 8 - Тест программы по методу Шеннона – Фано

На рис. 9 представлен тест программы для метода Хаффмана.

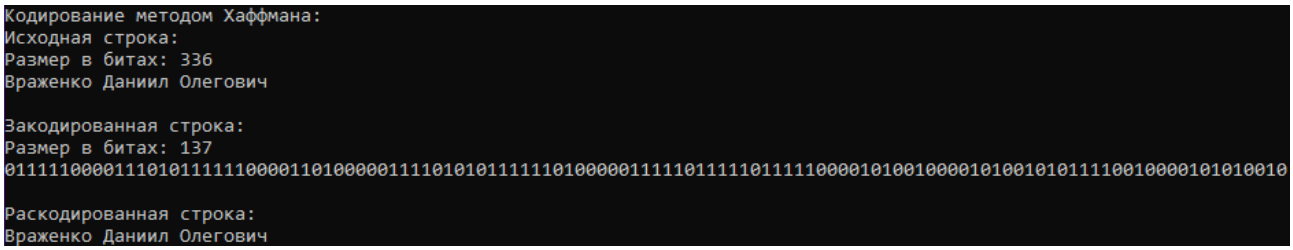

Рисунок 9 - Тест программы по методу Хаффманга

Часть 8.2. Реализация алгоритмов на основе сокращения числа переборов

Формулировка задачи:

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

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

Индивидуальный вариант: 6.

Задача: Дано прямоугольное поле размером n*m клеток. Можно совершать шаги длиной в одну клетку вправо, вниз или по диагонали вправо-вниз. В каждой клетке записано некоторое натуральное число. Необходимо попасть из верхней левой клетки в правую нижнюю. Вес маршрута – это сумма чисел всех посещенных клеток. Найти маршрут с минимальным весом.

Метод: Динамическое программирование.

Математическая модель решения (описание алгоритма):

Для переборного решения необходимо проверить все возможные пути из левой верхней клетки в правую нижнюю, чьё число возрастает очень быстро.

Для эффективного решения достаточно лишь каждый раз для каждой клетки выбирать из трех возможных чисел, откуда можно в неё попасть, наименьшее и прибавлять к нему собственное значение клетки.

Таким образом после применения алгоритма ко всем клеткам в после завершения работы алгоритма получим в правой нижней клетке минимальное возможное число.

Код программы с комментариями:

На рис. 10 представлен код функции для создания массива со случайными числами.

```
vector<vector<int>> generateRandomMatrix(int n, int m) // создание массива из случайных чисел
{
    random_device rd;
    mt19937 gen(rd());
    vector<vector<int>> arr(n, vector<int>(m, 0));
    for (int i = 0; i < n; ++i) // проход по строкам
        for (int j = 0; j < m; ++j) // проход по столбцам
            arr[i][j] = rd() % 1000; // заполнение ячейки случайным значением
    return arr;
}
```

Рисунок 10 - Код функции для создания массива со случайными числами

На рис. 11 представлен код функции для вывода массива на экран.

```
void print(vector<vector<int>>& arr) // вывод матрицы на экран
{
    for (int i = 0; i < arr.size(); ++i) // проход по строкам
    {
        for (int j = 0; j < arr[0].size(); ++j) // проход по столбцам
        {
            int spaces = 1;
            if (arr[i][j] > 0)
                spaces = log10(arr[i][j]);
            spaces = 10 - spaces;
            for (int k = 0; k < spaces; k++) // вывод проьелов на экран
                cout << ' ';
            cout << arr[i][j];
        }
        cout << '\n';
    }
}
```

Рисунок 11 - Код функции вывода массива на экран

На рис. 12 представлен код функции для нахождения количества всех возможных вариантов перемещений в массиве.

```
vector<vector<int>> countVariants(int n, int m) // метод для подсчета всех возможных вариантов движения
{
    vector<vector<int>> arr(n, vector<int>(m, 0)); // создание чистого массива
    arr[0][0] = 1;
    for (int i = 1; i < n; ++i) // проход по левой границе
        arr[i][0] += arr[i - 1][0];
    for (int j = 1; j < m; ++j) // проход по верхней границе
        arr[0][j] += arr[0][j - 1];
    for (int i = 1; i < n; ++i) // проход по внутреннему содержимому
        for (int j = 1; j < m; ++j)
            arr[i][j] += arr[i - 1][j] + arr[i][j - 1] + arr[i - 1][j - 1];
    return arr; // возврат массива с числами - количеством возможных путей к данной позиции
}
```

Рисунок 12 - Код функции для нахождения максимального количества путей

На рис. 13 представлен код функции для нахождения минимального числа между тремя числами.

```
int minimum(int a, int b, int c) // нахождение минимума среди трех чисел
{
    if (b < a) a = b;
    if (c < a) a = c;
    return a;
}
```

Рисунок 13 - Код для нахождения минимального числа между тремя числами

На рис. 14 представлен код функции для нахождения минимальной суммы в после прохождения массива из левой верхней ячейки в правую нижнюю.

```
vector<vector<int>> minSum(vector<vector<int>>& baseArr) // нахождение минимального пути в массиве
{
    int n = baseArr.size(); // вычисление кол-ва строк
    int m = baseArr[0].size(); // вычисление кол-ва столбцов
    vector<vector<int>> arr(n, vector<int>(m, 0)); // создание пустого массива
    arr[0][0] = baseArr[0][0]; // присваивание левой верхней ячейке нового массива значение левой верхней ячейки начального массива
    for (int i = 1; i < n; ++i) // проход по левой границе
        arr[i][0] = arr[i - 1][0] + baseArr[i][0];
    for (int j = 1; j < m; ++j) // проход по верхней границе
        arr[0][j] = arr[0][j - 1] + baseArr[0][j];
    for (int i = 1; i < n; ++i) // проход по внутренней части
        for (int j = 1; j < m; ++j)
            arr[i][j] = baseArr[i][j] + minimum(arr[i - 1][j], arr[i][j - 1], arr[i - 1][j - 1]);
    return arr;
}
```

Рисунок 14 - Код функции для нахождения минимальной суммы

На рис. 15 представлен код основной функции программы.

```
void ciaod_8_2()
{
    const int n = 4, m = 5;

    cout << "Исходная матрица:\n";
    vector<vector<int>> arr = generateRandomMatrix(n, m);
    print(arr);

    cout << "\nМатрица количества вариантов:\n";
    vector<vector<int>> newArr = countVariants(n, m);
    print(newArr);

    cout << "\nМатрица минимальных значений:\n";
    vector<vector<int>> minArr = minSum(arr);
    print(minArr);
}
```

Рисунок 15 - Код основной части программы

Результаты тестирования:

На рис. 16 представлен тест программы при n = 4, m = 5.

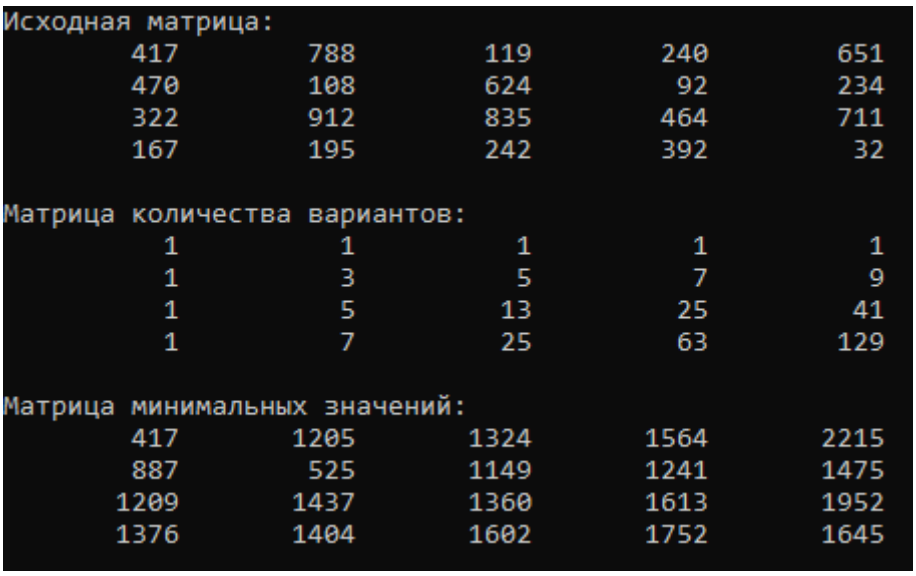

Рисунок 16 - Тест 1

На рис. 17 представлен также тест программы при n = 4, m = 5, но с другими значениями в массиве, потому что он заполняется случайными числами при каждом запуске.

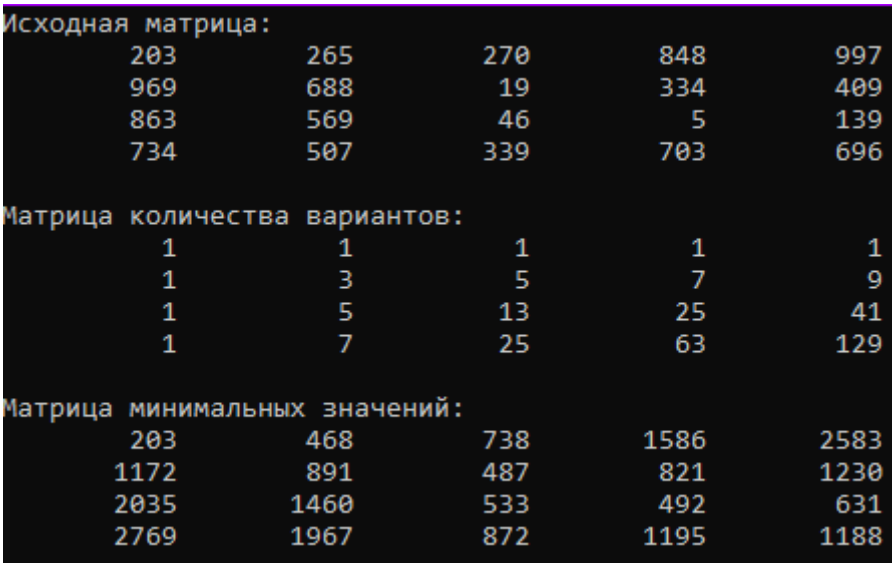

Рисунок 17 - Тест 2

На рис. 18 представлен тест программы при n = 7, m = 10.

```
Исходная матрица:
       930        564         61        389        813         82        652        415         42        149
       423        753        530        715        920        786         21        871        209        252
       886        496        593        926        987        201        705        152         11        615
       592        449        207         55        651        297        979        850        224        273
       459        698        752        131        986        903        877        187        579        386
        95        241        366        345        689        782        123        290        250        188
        52        282        437         51        711        574        984        609        662        194

Матрица количества вариантов:
         1          1          1          1          1          1          1          1          1          1
         1          3          5          7          9         11         13         15         17         19
         1          5         13         25         41         61         85        113        145        181
         1          7         25         63        129        231        377        575        833       1159
         1          9         41        129        321        681       1289       2241       3649       5641
         1         11         61        231        681       1683       3653       7183      13073      22363
         1         13         85        377       1289       3653       8989      19825      40081      75517

Матрица минимальных значений:
       930       1494       1555       1944       2757       2839       3491       3906       3948       4097
      1353       1683       2024       2270       2864       3543       2860       3731       3940       4192
      2239       1849       2276       2950       3257       3065       3565       3012       3023       3638
      2831       2298       2056       2111       2762       3059       4038       3862       3236       3296
      3290       2996       2808       2187       3097       3665       3936       4049       3815       3622
      3385       3237       3174       2532       2876       3658       3781       4071       4065       3810
      3437       3519       3611       2583       3243       3450       4434       4390       4727       4004
```

Рисунок 18 - Тест 3

На рис. 19 представлен тест программы при n = 5, m = 2.

```
Исходная матрица:
       327        128
       556        674
       911        871
       121        319
       102        373

Матрица количества вариантов:
         1          1
         1          3
         1          5
         1          7
         1          9

Матрица минимальных значений:
       327        455
       883       1001
      1794       1754
      1915       2073
      2017       2288
```

Рисунок 19 - Тест 4

ВЫВОД

Тексты часто содержат избыток информации, который можно устранить с помощью алгоритмов кодирования и сжатия. Эти методы позволяют значительно уменьшить объём данных, снижая нагрузку на хранилище или каналы передачи. Однако за это приходится платить временем на сжатие и последующую распаковку данных.

Во многих практических задачах, наряду с полным переборным подходом, существуют более эффективные алгоритмы. Они значительно сокращают вычислительные затраты, снижая сложность задачи и ускоряя её решение. Такие алгоритмы часто применяются в тех областях, где важны не только точность, но и время обработки данных. Эффективные методы позволяют обрабатывать огромные объёмы информации за приемлемое время, что особенно актуально в условиях быстро меняющегося мира технологий и данных.

Так, использование современных методов сжатия и оптимизации значительно улучшает производительность как в программных, так и в технических системах, облегчая работу с большими данными и ускоряя процессы вычислений.

СПИСОК ЛИТЕРАТУРЫ

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих, 2017. – С. 100-126.
2. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013. – С. 285-318.
3. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
4. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).
5. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).