



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 №8.1

Тема: «ОТДЕЛЬНЫЕ ВОПРОСЫ АЛГОРИТМИЗАЦИИ»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент
группа

Кузнецов Л. А.
ИКБО-20-23

Москва 2024

СОДЕРЖАНИЕ

ЧАСТЬ 8.1.....	3
Задание 1.....	3
Вариант.....	3
Решение.....	3
Задание 2.....	6
Решение.....	6
Тесты.....	10
ВЫВОД.....	13
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ.....	14

ЧАСТЬ 8.1

Задание 1

- 1) Выполнить сжатие элементов по методу Лемпеля-Зива LZ77 используя двухсимвольный алфавит.
- 2) Закодировать фразу, используя код LZ78.
- 3) Описать процесс сжатия/кодировки

Вариант

18 вариант.

Код для сжатия элементов по методу Лемпеля-Зива LZ77:

1110100110111001101.

Фраза для кодировки при помощи кода LZ78: sionsinossionsinos.

Фраза для кодировки методами Шеннона-Фано: “Дрынцы-брынцыбубенцы, Раззвонилисьсудальцы, Диги-диги-диги-дон, Выхо-ди-скорее-вон!”.

Решение

Начнём с сжатия элементов по методу Лемпеля-Зива LZ77.

Отообразим при помощи таблицы 1 ход присвоения кода для конкретной последовательности.

Таблица 1 – Ход присвоения последовательностям кода

Содержимое окна (сжимаемый текст)	Содержимое упреждающего буфера	Код назначенный последовательности
	1110100110111001101	
1	110100110111001101	1
11	0100110111001101	10
01	00110111001101	11
00	110111001101	100
110	111001101	101
111	001101	110
001	101	111
101		1000

Теперь же перейдём к составлению конечной последовательности как символов, так и кодов.

В первые две очереди последовательности кодов запишем сначала 1, а после 11, так как обе эти комбинации уникальны

Послед-ость: 1.11.

Теперь присвоим последовательности комбинацию 01 – она уникальна, а её код длины 2, поэтому записываем её без изменений.

Послед-ость: 1.11.01.

Далее идёт комбинация 00 с кодом 100. Комбинация уникальна, но её код длины 3, поэтому увеличиваем длину 00 до 000 и записываем полученную комбинацию в последовательность.

Послед-ость: 1.11.01.000.

После переходим к комбинации 110 – она не уникальна, поэтому присваиваем ей код 10 от комбинации 11 и дописываем 0, чтобы длина полученной комбинации была 3

Послед-ость: 1.11.01.000.100.

Далее следует 111, которой присваивается код 10 от 11 и дописывается 1 в конце.

Послед-ость: 1.11.01.000.100.101.

Рассмотрим комбинацию 001. Сначала присвоим ей код 100 от 00, а после допишем 1.

Послед-ость: 1.11.01.000.100.101.1001.

И последний элемент – 101. Он уникален, но из-за того, что длина предыдущей последовательности 4, то и эта последовательность должна иметь не меньшую длину. Тогда в итоге получим 0101.

Послед-ость: 1.11.01.000.100.101.1001.0101.

В конце всех преобразований мы получили следующие последовательность и её код.

Послед-ость: 1.11.01.000.100.101.1001.0101

Её код: 1.10.11.100.101.110.111.1000

Теперь же перейдём к кодированию фразы при помощи кода LZ78.

Фраза: sionsinossionsinos

Отообразим результаты кодировки при помощи таблицы 2.

Таблица 2 – Процесс кодировки фразы методом LZ78

Словарь	Считываемое содержимое	Код
	s	<0, s>
s = 1	i	<0, i>
s = 1, i = 2	o	<0, o>
s = 1, i = 2, o = 3	n	<0, n>
s = 1, i = 2, o = 3, n = 4	si	<1, i>
s = 1, i = 2, o = 3, n = 4 si = 5	no	<4, o>
s = 1, i = 2, o = 3, n = 4 si = 5, no = 6	ss	<1, s>
s = 1, i = 2, o = 3, n = 4 si = 5, no = 6, ss = 7	io	<2, o>
s = 1, i = 2, o = 3, n = 4 si = 5, no = 6, ss = 7, io = 8	ns	<4, s>
s = 1, i = 2, o = 3, n = 4 si = 5, no = 6, ss = 7, io = 8 ns = 9	in	<2, n>
s = 1, i = 2, o = 3, n = 4 si = 5, no = 6, ss = 7, io = 8 ns = 9, in = 10	os	<3, s>
s = 1, i = 2, o = 3, n = 4 si = 5, no = 6, ss = 7, io = 8 ns = 9, in = 10		<0, EOF>

Как видно из таблицы 2, мы постепенно проходились по фразе, убирая из неё уже увиденные комбинации, и добавляли в словарь неповторяющиеся.

Итоговая послед-ость: <0, s>, <0, i>, <0, o>, <0, n>, <1, i>, <4, o>, <1, s>, <2, o>, <4, s>, <2, n>, <3, s>, <0, EOF>.

Задание 2

- 1) Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона-Фано.
- 2) Описать разработанные методы и провести тестирование.

Решение

Для реализации метода Шеннона-Фано сначала посчитаем кол-во каждого из символов, после отсортируем полученные значения по невозрастанию, затем делим полученные значения по частям и строим дерево с символами и выводим его на экран.

После выяснения стратегии решения задачи был написан код.

В первую очередь была разработана логика структуры dict, которая хранит в себе значения кол-ва повторений конкретного символа и выполняет работу словаря (рис. 1-3).

```
struct dict {
    int size = 1;
    char letter;
    int count=1;
    string path = "";
    dict* next_element=nullptr;
    dict* prev_element = nullptr;
    dict(char _letter) : letter(_letter) {}

    dict* find(char _letter) {
        if (this->letter == _letter)
            return this;
        else
        {
            if (this->next_element == nullptr)
                return nullptr;
            else
                return next_element->find(_letter);
        }
    }

    dict* find(int pos) {
        if (pos != 0)
            if (next_element == nullptr)
                return nullptr;
            else
                return next_element->find(pos - 1);
        else
            return this;
    }

    void add(dict* new_element) {
        this->size += 1;
        if (this->next_element == nullptr)
        {
            this->next_element = new_element;
            this->next_element->prev_element = this;
        }
        else
            this->next_element->add(new_element);
    }

    void sort(dict* head, int st=0) {
        if (this->next_element != nullptr)
        {
            if (this->next_element->count > this->count)
            {
                int temp_st = st;
                dict* temp_dict = this;
                while (temp_dict->prev_element != nullptr)
                {
                    if (this->next_element->count > temp_dict->prev_element->count)
                    {
                        temp_dict = temp_dict->prev_element;
                        temp_st--;
                    }
                    else
                        break;
                }
                if (temp_st != 0) {
```

Рисунок 1 – 1-ая часть словаря dict

```

64         if (temp_st != 0) {
65             this->next_element->prev_element = temp_dict->prev_element;
66             temp_dict->prev_element->next_element = this->next_element;
67             this->next_element = this->next_element->next_element;
68             this->next_element->prev_element = this;
69             temp_dict->prev_element = temp_dict->prev_element->next_element;
70             temp_dict->prev_element->next_element = temp_dict;
71         }
72         else
73         {
74             head = this->next_element;
75             this->next_element->size = temp_dict->size;
76             temp_dict->prev_element = this->next_element;
77             this->next_element = this->next_element->next_element;
78             temp_dict->prev_element->next_element = temp_dict;
79             this->next_element->prev_element = this;
80             temp_dict->prev_element->prev_element = nullptr;
81         }
82         this->sort(head, st + 1);
83     }
84     else
85         this->next_element->sort(head, st + 1);
86 }
87 else
88     return;
89 }
90
91 void out() {
92     int pos = 0;
93     dict* temp_slovarb = this->find(pos);
94     while (temp_slovarb != nullptr)
95     {
96         cout << temp_slovarb->letter << " - " << temp_slovarb->count << " - " << temp_slovarb->path << endl;
97         pos++;
98         temp_slovarb = this->find(pos);
99     }
100     cout << "-----" << endl;
101 }
102
103 int sum(int st, int end) {
104     int summ = 0;
105     while (st != end)
106     {
107         summ += this->find(st)->count;
108         st++;
109     }
110     return summ;
111 }
112
113 void make_path(int st, int end) {
114     int summ = 0;
115     int end_pos = 0;
116     if (end - st > 1)
117     {
118         bool phase_of_zeroes = true;
119         for (int i = st; i < end; i++)
120         {
121             if (phase_of_zeroes)
122             {
123                 this->find(i)->path += "0";

```

Рисунок 2 – 2-ая часть словаря dict

```

        this->find(i)->path += "0";
        summ += this->find(i)->count;
        if (summ >= (this->sum(st, end) / 2) || ((summ + this->find(i+1)->count) >= this->sum(i+1, end)))
        {
            phase_of_zeroes = false;
            end_pos = i + 1;
            this->make_path(st, end_pos);
        }
    }
    else
        this->find(i)->path += "1";
    this->make_path(end_pos, end);
}

void sort_path() {
    dict* temp_main_dict = this;
    dict* temp_secondary_dict;
    string temp_path;
    while (temp_main_dict->next_element != nullptr)
    {
        temp_secondary_dict = temp_main_dict->next_element;
        while (temp_secondary_dict != nullptr)
        {
            if (temp_main_dict->path.size() > temp_secondary_dict->path.size())
            {
                temp_path = temp_main_dict->path;
                temp_main_dict->path = temp_secondary_dict->path;
                temp_secondary_dict->path = temp_path;
            }
            temp_secondary_dict = temp_secondary_dict->next_element;
        }
        temp_main_dict = temp_main_dict->next_element;
    }
}

int count_bytes() {
    dict* temp_dict = this;
    int summ = 0;
    while (temp_dict != nullptr)
    {
        summ += temp_dict->count * temp_dict->path.length();
        temp_dict = temp_dict->next_element;
    }
    return summ;
}
};

```

Рисунок 3 – 3-я часть словаря dict

Данный словарь реализует все необходимые взаимодействия с символами: создаёт путь для дерева, хранит значения повторений, а также ссылку на следующий элемент, осуществляет поиск элементов по индексу и символу, совершает вывод данных на экран, а также находит сумму бит всех данных.

Как оказывается, прописка логики словаря – это самая сложная и интересная часть данной работы.

Далее последовала реализация логики листьев будущего дерева, а именно структуры leaf (рис. 4).

```
struct leaf {  
    string value = "";  
    leaf* right_leaf = nullptr;  
    leaf* left_leaf = nullptr;  
    leaf* head = nullptr;  
  
    void add_dict(dict* _dict) {  
        leaf* temp_leaf = this;  
        for (char letter : _dict->path) {  
            if (letter == '0')  
            {  
                if (temp_leaf->left_leaf == nullptr)  
                {  
                    temp_leaf->left_leaf = new leaf("0");  
                    temp_leaf->left_leaf->head = temp_leaf;  
                }  
                temp_leaf = temp_leaf->left_leaf;  
            }  
            else  
            {  
                if (temp_leaf->right_leaf == nullptr)  
                {  
                    temp_leaf->right_leaf = new leaf("1");  
                    temp_leaf->right_leaf->head = temp_leaf;  
                }  
                temp_leaf = temp_leaf->right_leaf;  
            }  
        }  
        temp_leaf->value = temp_leaf->value + " " + _dict->letter;  
    }  
  
    leaf(string _value) : value(_value) {};  
  
    void out(string text, bool right=true) {  
        if (right)  
            cout << text << "-" << this->value << endl;  
        else  
            cout << text << "--" << this->value << endl;  
  
        if (this->right_leaf != nullptr)  
            this->right_leaf->out(text + "\t|");  
  
        if (this->left_leaf != nullptr)  
            this->left_leaf->out(text + "\t", false);  
    }  
};
```

Рисунок 4 – Реализация логики структуры leaf

Данные листья по своей сути и являются самым деревом. Что имеется под этим ввиду? Имеется ввиду то, что дерево заранее не определено и оно формируется в зависимости от путей элементов структуры dict, которая впоследствии задаёт само дерево через структуру leaf.

И последней частью реализации работы программы в соответствии с заданными алгоритмами является реализация работы функции main (рис. 5, 6).

```
int main() {
    setlocale(LC_ALL, "Russian");

    string text = "";
    int choice;

    cout << "Какая часть задания (1 или 2)?" << endl;
    cin >> choice;
    if (choice == 1)
    {
        text = "Дрынцы-брынцыбубенцы, Раззвонилисьсудальцы, Диги-диги-диги-дон, Выхо-ди-скорее-вон!";
        dict* slovarb = new dict(char(tolower(text[0])));
        dict* temp_slovarb;
        slovarb->find(char(tolower(text[0]))->count--;
        for (char letter : text)
        {
            letter = tolower(letter);
            temp_slovarb = slovarb->find(letter);
            if (temp_slovarb != nullptr)
                temp_slovarb->count++;
            else
                slovarb->add(new dict(letter));
        }
        slovarb->out();
        slovarb->sort(slovarb);
        while (slovarb->prev_element != nullptr)
            slovarb = slovarb->prev_element;
        slovarb->make_path(0, slovarb->size);
        slovarb->out();
        cout << slovarb->count_bytes() << endl;
        slovarb->sort_path();
        slovarb->out();
        cout << slovarb->count_bytes() << endl;

        leaf* body = new leaf("head");
        for (int i = 0; i < slovarb->size; i++)
            body->add_dict(slovarb->find(i));

        body->out("");
    }
}
```

Рисунок 5 – 1-ая часть реализации алгоритма работы функции main

В первой части представлена работы программы с методом Шеннона-Фано, а во втором – методом Хаффмана.

```

    }
    else
    {
        text = "Кузнецов Лев Андреевич";
        dict* slovarb = new dict(char(tolower(text[0]]));
        dict* temp_slovarb;
        slovarb->find(char(tolower(text[0]))->count--;
        for (char letter : text)
        {
            letter = tolower(letter);
            temp_slovarb = slovarb->find(letter);
            if (temp_slovarb != nullptr)
                temp_slovarb->count++;
            else
                slovarb->add(new dict(letter));
        }
        slovarb->out();
        slovarb->sort(slovarb);
        while (slovarb->prev_element != nullptr)
            slovarb = slovarb->prev_element;
        slovarb->out();
        string path_text = "";
        for (int i = 0; i < (slovarb->size-1); i++)
        {
            slovarb->find(i)->path=path_text + "0";
            path_text += "1";
        }
        slovarb->find(slovarb->size-1)->path = path_text;
        slovarb->out();

        leaf* body = new leaf("head");
        for (int i = 0; i < slovarb->size; i++)
            body->add_dict(slovarb->find(i));

        body->out("");
    }

    return 0;
}

```

Рисунок 6 – 2-ая часть реализации алгоритма работы функции main

Как можно заметить программа работает с текстом и кодирует его в соответствии с заданным текстом, что и требовалось выполнить.

Теперь настало время провести необходимые тесты.

Тесты

Для начала проведём тесты с алгоритмом Шеннона-Фано.

Фраза была описана выше.

Начнём тест с того, что подсчитаем все значения повторений элементов, выведем сжатый и первоначальный вариант кол-ва битов. (рис. 7)

На рисунке 8 видно, что корень дерева отмечен как head, чтобы было проще сориентироваться, далее к каждому листу присоединено его значение пути (то есть 1 или 0), а также символ, которому соответствует этот путь.

Составим таблицу 3 вероятностей для данного случая.

Таблица 3 – Таблица вероятности появления символа в тексте

Алфавит	и	д	ы	-	н	о	р	ц
Кол. вх.	9	7	7	7	6	5	4	4
Вероятн.	0,11	0,08	0,08	0,08	0,07	0,06	0,05	0,05

Алфавит	б	е	,	« »	в	г	у	а
Кол. вх.	3	3	3	3	3	3	2	2
Вероятн.	0,04	0,04	0,04	0,04	0,04	0,04	0,02	0,02

Алфавит	з	л	с	ь	х	к	!
Кол. вх.	2	2	2	2	1	1	1
Вероятн.	0,02	0,02	0,02	0,02	0,01	0,01	0,01

Теперь проведём тесты с методом Хаффмана (рис. 9).

```

е - 4 - 0
в - 3 - 10
н - 2 - 110
- 2 - 1110
к - 1 - 11110
у - 1 - 111110
з - 1 - 1111110
ц - 1 - 11111110
о - 1 - 111111110
л - 1 - 1111111110
а - 1 - 11111111110
д - 1 - 111111111110
р - 1 - 1111111111110
и - 1 - 11111111111110
ч - 1 - 11111111111111
-----
133
176

```

Рисунок 9 – 1-ая часть результатов тестов метода Хаффмана

Как видно из тестов, текст не сильно-то и сжался, однако алгоритм реализации подобного метода является чрезвычайно простым в исполнении.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
2. Документация по языку C++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ruru/cpp/cpp/> (дата обращения 01.09.2021).
3. 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).