МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА» - Российский технологический университет»

РТУ МИРЭА

Отчёт по выполнению практического задания № 8.2
Тема:
«Алгоритмы кодирования и сжатия данных»

Выполнил студент: Данов Арсений Иванович

Группа: ИКБО-10-23

Москва 2024

Вариант:

Задание 1:

Номер: 6

Закодировать фразу методами Шеннона–Фано:

По-турецки говорили. Чяби, чяряби Чяряби, чяби-чяби. Мы набрали в рот воды.

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

000101110110100111

Закодировать следующую фразу, используя код LZ78:

менменаменаменатеп

Задание 2:

Провести кодирование(сжатие) исходной строки символов «Данов Арсений Иванович» с использованием алгоритма Хаффмана.

1.1 Формулировка задачи

Закодировать фразу методами Шеннона–Фано.

1.2 Ход решения

Фраза: По-турецки говорили. Чяби, чяряби Чяряби, чяби-чяби. Мы набрали в рот воды.

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	Бит
« »	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
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: 75 * 8 бит = 600 бит

Закодированная фраза: 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.

2.1 Формулировка задачи

Сжатие данных по методу Лемпеля–Зива LZ77.

2.2 Ход решения

Фраза: 0001010010101001101

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

3.1 Формулировка задачи

Закодировать фразу, используя код LZ78

3.2 Ход решения

Фраза: менменаменаменатеп

Словарь	Считываемое	Код
	м	<0, “м”>
1: м;	е	<0, “е”>
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; 5: на; 2: е; 3: н; 4: ме;	мен	<4, “н”>
1: м; 5: на; 2: е; 6: мен; 3: н; 4: ме;	а	<0, “а”>
1: м; 5: на; 2: е; 6: мен; 3: н; 7: а; 4: ме;	мена	<6, “а”>
1: м; 5: на; 2: е; 6: мен; 3: н; 7: а; 4: ме; 8: мена;	т	<0, “т”>
1: м; 5: на; 9: т 2: е; 6: мен; 3: н; 7: а; 4: ме; 8: мена;	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

4.1 Формулировка задачи

Разработать программы сжатия и восстановления текста методом Шеннона — Фано.

4.2 Ход решения

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

4.3 Исходный код программы

На рисунках 1-3 представлены исходные коды структуры, а также функций создания словаря, кодирования и декодирования.

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

bool compare_lets(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 — Исходный код программы

```cpp
map<char, string> make_map(string text)
{
    map<char, string> codes;
    vector<let *> counts;

    string set_text = "";
    for (int i = 0; i < text.length(); ++i)
    {
        if (count(set_text.begin(), set_text.end(), text[i]) == 0)
        {
            set_text += text[i];
            codes[text[i]] = "";
        }
    }

    for (int i = 0; i < set_text.length(); ++i)
    {
        counts.push_back(new let(set_text[i], count(text.begin(), text.end(), set_text[i])));
    }

    sort(counts.begin(), counts.end(), compare_lets);

    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 — Исходный код программы

```
90  string code(string text, map<char, string> codes)
91  {
92      string s = "";
93      for (int i = 0; i < text.length(); ++i)
94      {
95          s += codes[text[i]];
96      }
97      return s;
98  }
99
100 string decode(string text, map<char, string> codes)
101 {
102     string s = "";
103     string buffer = "";
104     for (int i = 0; i < text.length(); ++i)
105     {
106         buffer += text[i];
107         for (auto &[lt, code] : codes)
108         {
109             if (code == buffer)
110             {
111                 s += lt;
112                 buffer = "";
113                 break;
114             }
115         }
116     }
117     return s;
118 }
```

Рисунок 3 — Исходный код программы

4.4 Результаты тестирования

На рисунке 4-5 представлены результаты тестирования программы.

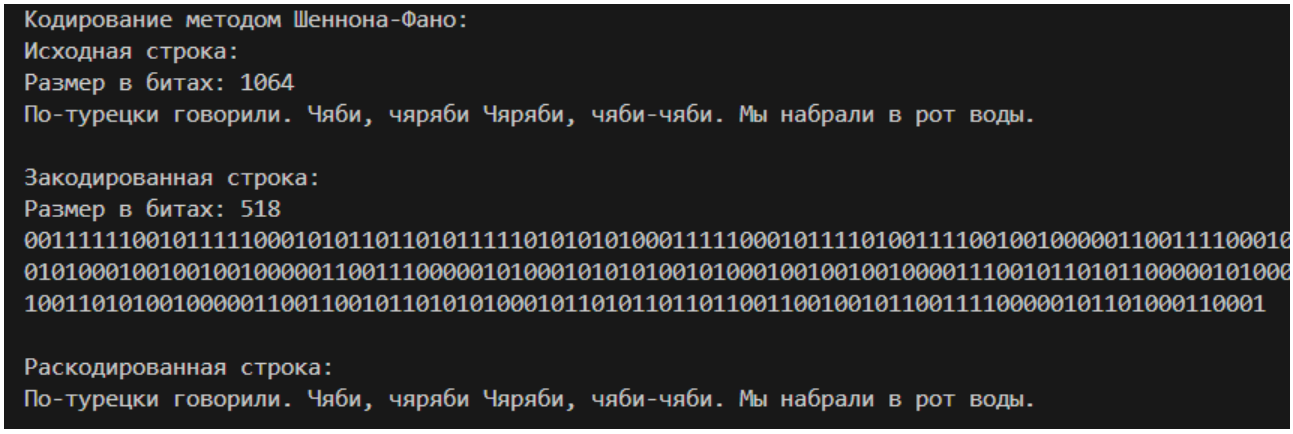

Рисунок 4 — Результаты тестирования программы

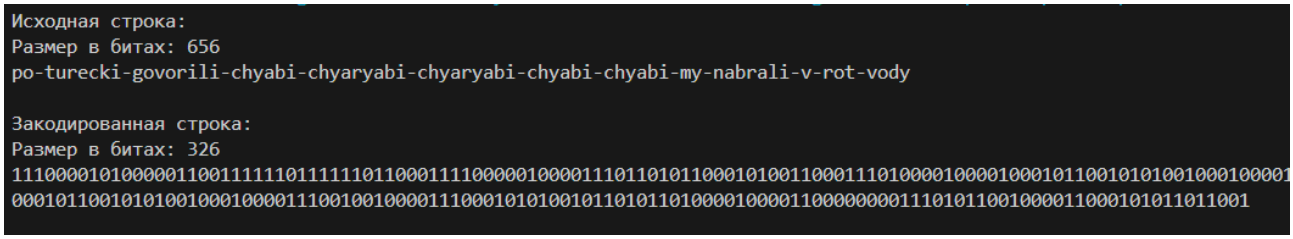

Рисунок 5 — Результаты тестирования программы

5.1 Формулировка задачи

Разработать программы сжатия и восстановления текста методом Хаффмана.

5.2 Ход решения

Первым делом формируется структура данных, хранящая коды для каждого встречающегося символа. Согласно алгоритму считается количество вхождений каждого символа, после чего символы сортируются в порядке неубывания их количеств вхождений в строку.

Два самых редких символа объединяются одним родителем, чьё количество повторений равно сумме объединенных символов. Родитель добавляется в общий список, а листья удаляются, после чего алгоритм повторяется. Меньшему листу присваивается код 0, большему — 1.

Условием окончания алгоритма будет являться оставшийся лишь один общий родитель, также являющийся деревом-словарём.

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.

5.3 Исходный код программы

На рисунках 6-8 представлены исходные коды структуры, а также функций создания словаря, кодирования и декодирования.

```
120  struct haf
121  {
122      char lt;
123      char code;
124      int cnt;
125      haf *left;
126      haf *right;
127      haf(char lt, int cnt, haf *left, haf *right, char code)
128      {
129          this->lt = lt;
130          if (left && right)
131          {
132              this->cnt = left->cnt + right->cnt;
133          }
134          else
135          {
136              this->cnt = cnt;
137          }
138          this->left = left;
139          this->right = right;
140          this->code = code;
141      }
142      char find(string code, int ind = 0)
143      {
144          if (!this->left && !this->right)
145          {
146              if (code.length() == ind)
147              {
148                  return this->lt;
149              }
150              else
151              {
152                  return '@';
153              }
154          }
155
156          if (ind >= code.length())
157          {
158              return '@';
159          }
160
161          if (code[ind] == '0')
162          {
163              return this->left->find(code, ind + 1);
164          }
165          return this->right->find(code, ind + 1);
166      }
167
```

Рисунок 6 — Исходный код программы

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

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

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

haf *make_haf(string text)
{
    vector<haf *> cnts;

    string set_text = "";
    for (int i = 0; i < text.length(); ++i)
    {
        if (count(set_text.begin(), set_text.end(), text[i]) == 0)
        {
            set_text += text[i];
        }
    }

    for (int i = 0; i < set_text.length(); ++i)
    {
        cnts.push_back(new haf(set_text[i], count(text.begin(), text.end(), set_text[i]), nullptr, nullptr, '@'));
    }
    haf *ptr;
    while (cnts.size() > 1)
    {
        sort(cnts.begin(), cnts.end(), compare_hafs);
        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 7 — Исходный код программы

```
223  string code_h(string text, haf* codes)
224  {
225      string s = "";
226      for (int i = 0; i < text.length(); ++i)
227      {
228          s += codes->get_code(text[i]);
229      }
230      return s;
231  }
232
233  string decode_h(string text, haf* codes)
234  {
235      string s = "";
236      string buffer = "";
237      char lt = '@';
238      for (int i = 0; i < text.length(); ++i)
239      {
240          buffer += text[i];
241          lt = codes->find(buffer);
242          if (lt != '@') {
243              s += lt;
244              buffer = "";
245          }
246      }
247      return s;
248  }
```

Рисунок 8 — Исходный код программы

5.4 Результаты тестирования

На рисунке 9-10 представлены результаты тестирования программы.

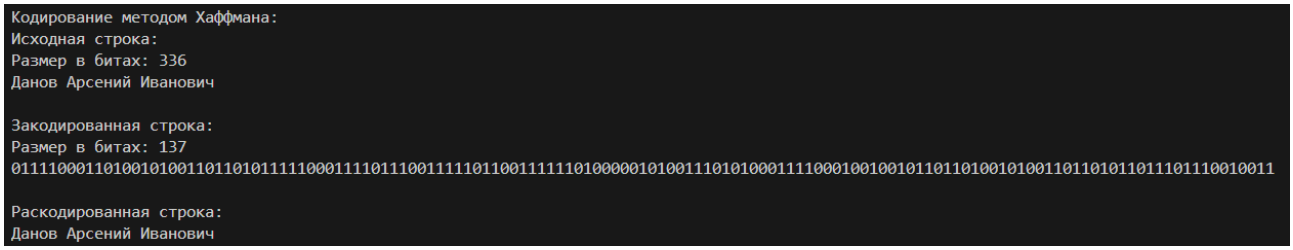

Рисунок 4 — Результаты тестирования программы

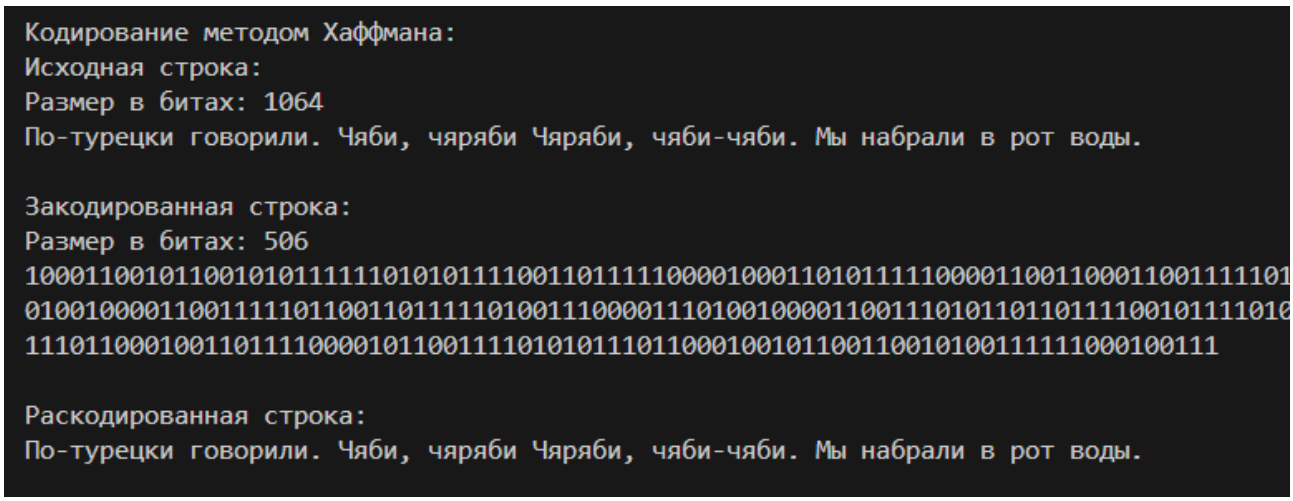

Рисунок 5 — Результаты тестирования программы

6.1 Выводы

Зачастую в представлении текстова присутствует избыточная информация, сокращать которую призваны алгоритмы кодирования и сжатия текста. Это позволяет сильно сокращать занимаемый информацией объём, жертвуя при этом временем на сжатие и распаковку.