

Индивидуальное задание №4

Проектирование и реализация нормального алгоритма Маркова

Патычкина Елизавета Вадимовна, КТб01-10

Вариант 3.22

1. Постановка задачи

Входное слово НАМ представляет собой целое неотрицательное двоичное число. Получить восьмеричное представление этого числа. Незначащих нулей ни во входном, ни в выходном числе нет.

2. Словесное описание алгоритма решения задачи с помощью нормального алгоритма Маркова

Программа вводит маркер в начало строки с исходным целым неотрицательным двоичным числом. Далее двигает маркер в конец строчки (Например: начальное положение маркера: *1001, конечное положение маркера: 1001*). Как только маркер доходит до конца строки меняем вид маркера (Например: 1001* → 1001!). Далее с конца переводим из двоичного числа в восьмеричное по три символа и двигаем текущий маркер вперед последнего переведенного числа до тех пор, пока маркер не достиг начала строки. В случае если не хватает символов для перевода, мысленно дописываем нули впереди числа и выполняем перевод (Например, строка “10!234” мысленно допишет один ноль и будет представлена как “!2234”). Как только текущий маркер достиг начала строки, удаляем маркер.

3. Используемый алфавит ленты

$A = \{0, 1, 2, 3, 4, 5, 6, 7, *, !, ' '\}$ – алфавит нормального алгоритма Маркова (входной, выходной, вспомогательный)

$A_{\text{вх.}} = \{0, 1\}$ – входной алфавит нормального алгоритма Маркова

$A_{\text{вых.}} = \{0, 1, 2, 3, 4, 5, 6, 7\}$ – выходной алфавит нормального

алгоритма Маркова

$A_{\text{всп.}} = \{!, *, ', \prime\}$ – вспомогательный алфавит нормального алгоритма Маркова

4. Система подстановок нормального алгоритма

1. $*0 \rightarrow 0*$ - Двигаем метку в конец строки
2. $*1 \rightarrow 1*$ - Двигаем метку в конец строки
3. $* \rightarrow !$ - Меняем метку
4. $000! \rightarrow !0$ – С конца переводим из двоичной системы счисления в восьмеричную
5. $001! \rightarrow !1$ – С конца переводим из двоичной системы счисления в восьмеричную
6. $010! \rightarrow !2$ – С конца переводим из двоичной системы счисления в восьмеричную
7. $011! \rightarrow !3$ – С конца переводим из двоичной системы счисления в восьмеричную
8. $100! \rightarrow !4$ – С конца переводим из двоичной системы счисления в восьмеричную
9. $101! \rightarrow !5$ – С конца переводим из двоичной системы счисления в восьмеричную
10. $110! \rightarrow !6$ – С конца переводим из двоичной системы счисления в восьмеричную
11. $111! \rightarrow !7$ – С конца переводим из двоичной системы счисления в восьмеричную
12. $10! \rightarrow !2$ – Переводим из двоичной системы счисления в восьмеричную первое число без незначащих нулей

13. 11! \rightarrow !3 – Переводим из двоичной системы счисления в восьмеричную первое число без незначащих нулей

14. 1! \rightarrow !1 – Переводим из двоичной системы счисления в восьмеричную первое число без незначащих нулей

15. ! \rightarrow . – Завершаем работу

16. \rightarrow * – Вводим маркер

5. Набор тестов, охватывающих все режимы работы алгоритма и особые случаи

1) 1000100

Входное слово: 1000100

Выходное слово: 104

2) 10011001

Входное слово: 10011001

Выходное слово: 231

3) 11101010

Входное слово: 11101010

Выходное слово: 352

4) 0111

Входное слово: 0111

Выходное слово: Неверный ввод

5) 12101

Входное слово: 12101

Выходное слово: Неверный ввод

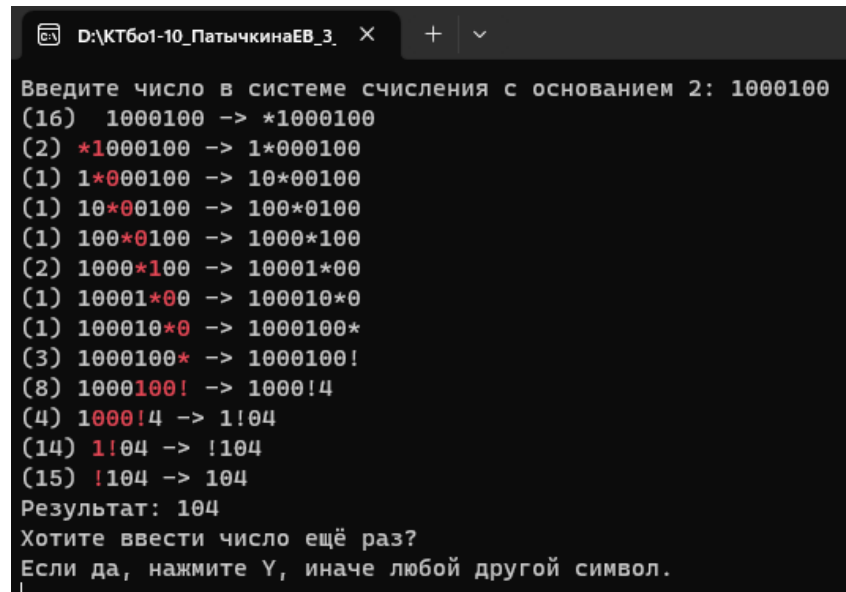
6) 11111110

Входное слово: 11111110

Выходное слово: 776

6. Скриншоты выполнения программы

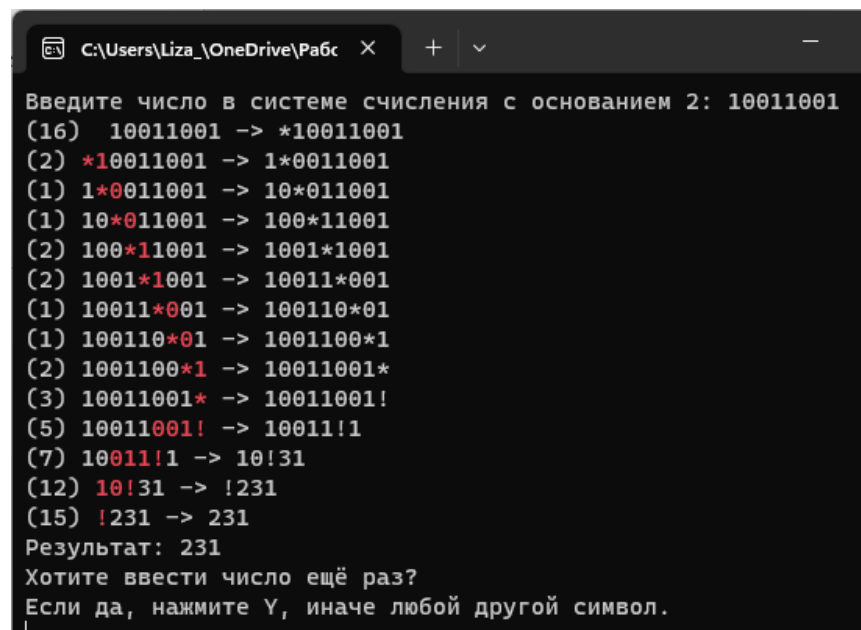
1) Рисунок 1 – Тест 1



```
D:\КТб01-10_ПатычкинаЕВ_3
Введите число в системе счисления с основанием 2: 1000100
(16) 1000100 -> *1000100
(2) *1000100 -> 1*000100
(1) 1*000100 -> 10*00100
(1) 10*00100 -> 100*0100
(1) 100*0100 -> 1000*100
(2) 1000*100 -> 10001*00
(1) 10001*00 -> 100010*0
(1) 100010*0 -> 1000100*
(3) 1000100* -> 1000100!
(8) 1000100! -> 1000!4
(4) 1000!4 -> 1!04
(14) 1!04 -> !104
(15) !104 -> 104
Результат: 104
Хотите ввести число ещё раз?
Если да, нажмите Y, иначе любой другой символ.
```

Рисунок 1 – Тест 1

2) Рисунок 2 – Тест 2



```
C:\Users\Liza_OneDrive\Рабочий стол
Введите число в системе счисления с основанием 2: 10011001
(16) 10011001 -> *10011001
(2) *10011001 -> 1*0011001
(1) 1*0011001 -> 10*011001
(1) 10*011001 -> 100*11001
(2) 100*11001 -> 1001*1001
(2) 1001*1001 -> 10011*001
(1) 10011*001 -> 100110*01
(1) 100110*01 -> 1001100*1
(2) 1001100*1 -> 10011001*
(3) 10011001* -> 10011001!
(5) 10011001! -> 10011!1
(7) 10011!1 -> 10!31
(12) 10!31 -> !231
(15) !231 -> 231
Результат: 231
Хотите ввести число ещё раз?
Если да, нажмите Y, иначе любой другой символ.
```

Рисунок 2 – Тест 2

3) Рисунок 3 – Тест 3

Рисунок 3 – Тест 3

4) Рисунок 4 – Тест 4

Рисунок 4 – Тест 4

5) Рисунок 5 – Тест 5

Рисунок 5 – Тест 5

6) Рисунок 6 – Тест 6

```
D:\КТб01-10_ПатычкинаЕВ_3_ X + v
Введите число в системе счисления с основанием 2: 11111110
(16) 11111110 -> *11111110
(2) *11111110 -> 1*11111110
(2) 1*11111110 -> 11*11111110
(2) 11*11111110 -> 111*11111110
(2) 111*11111110 -> 1111*11111110
(2) 1111*11111110 -> 11111*11111110
(2) 11111*11111110 -> 111111*11111110
(2) 111111*11111110 -> 1111111*11111110
(2) 1111111*11111110 -> 11111111*11111110
(2) 11111111*11111110 -> 111111111*11111110
(2) 111111111*11111110 -> 1111111111*11111110
(1) 1111111111*11111110 -> 11111111111*11111110
(3) 11111111111*11111110 -> 111111111111*11111110
(10) 111111111111*11111110 -> 1111111111111*11111110
(11) 1111111111111*11111110 -> 11111111111111*11111110
(11) 11111111111111*11111110 -> 111111111111111*11111110
(15) 111111111111111*11111110 -> 1111111111111111*11111110
Результат: 776
Хотите ввести число ещё раз?
Если да, нажмите Y, иначе любой другой символ.
```

Рисунок 6 – Тест 6

7. Описание структуры данных, используемой в программе для хранения системы подстановок

Для хранения одной подстановки используется структура substitution:

```
struct substitution
{
    int number;
    string start;
    string finish;
    bool stop;
};
```

Целочисленная переменная `number` хранит порядок подстановки в системе подстановок. Переменная `start` типа `string` хранит строку, которую необходимо заменить, или первую часть в подстановке. Переменная `finish` типа `string` хранит строку, на которую необходимо заменить часть исходной. Переменная `stop` типа `bool` хранит булевское значение характеристики является ли данная подстановка завершающей.

Система подстановок в данной программе представлена динамическим массивом, элементами которого являются структуры `substitution`.

8. Словесное описание программной реализации одного шага алгоритма

Идея программной реализации одного шага алгоритма заключается в следующем:

1. Функция принимает структуру хранения строки, в которой ранится сама строка, номер символа первой замены, логическая переменная, отвечающая за значение последней замены.
2. Функция перебирает все подстановки по порядку пока не нашлась подходящая.
3. В случае если нашлась первая подходящая подстановка, мы меняем флаг, чтобы не перебирать оставшиеся подстановки, меняем значение логической переменной, отвечающей за значение последней замены, на соответствующее значение в системе подстановок.
4. Далее перебираем каждую подстроку исходного слова и ищем порядок первого символа замены.
5. Выводим необходимые данные.
6. Меняем исходную подстроку, которую нашли в строке на ту что указана в системе постановок при помощи метода replace.
7. Выводим строку в измененном виде.
8. Если подстановка не является завершающей, то функция выполняется снова.

9. Листинг программы

```
//ЮФУ, ИКТИБ, МОП ЭВМ  
//Программирование и основы теории алгоритмов  
//Индивидуальное задание №4  
//Проектирование и реализация нормального алгоритма Маркова  
//КТб01-10, Патычкина Елизавета Вадимовна
```

```
#include <iostream>
```

```

#include <string>
#include <vector>
#include <Windows.h>
using namespace std;

HANDLE hConsole;

//Структура хранения команды из системы команд
struct substitution
{
    int number;//Номер команды
    string start;//Первая часть подстановки
    string finish;//Вторая часть подстановки
    bool stop;////Переменная, хранящая конец работы нормального алгоритма Маркова
};

//Структура хранения нормального алгоритма Маркова
struct n_a_m
{
    string line;//Текущая строка
    int position;//Номер позиции первого символа первой встречающейся подстановки
    bool stop;////Переменная, хранящая конец работы нормального алгоритма Маркова
};

//Входные параметры: string line - входная строка, введенная пользователем
//Функция: проверяет корректность введенной строки согласно требованиям в условии
//Выходные параметры: функция возвращает true, если введенная строка корректна
//
bool check(string line);

//Входные параметры: vector <struct substitution> ss - система подстановок
//Функция: инициализирует систему подстановок
//Выходные параметры: функция ничего не возвращает
void insert_substitution_system(vector <struct substitution>& ss);

//Входные параметры: string line - входная строка, введенная пользователем
//      struct n_a_m nam - структура хранения нормального алгоритма Маркова
//Функция: выбирает и реализуют подстановки для текущего слова и выводит результат шага
//Выходные параметры: функция ничего не возвращает
void function_choice_realization(vector <struct substitution>& ss, struct n_a_m& nam);

int main()
{
    setlocale(LC_ALL, "Russian");
    string line, answer;
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    do
    {
        cout << "Введите число в системе счисления с основанием 2: ";
        cin >> line;
        if (check(line) == false)
        {
            cout << "Число введено неверно." << endl;
        }
        else
        {
            vector <struct substitution> substitution_system;
            struct n_a_m NAM;
            NAM.line = " "+line;
            NAM.stop = false;
            insert_substitution_system(substitution_system);
            while (!NAM.stop)

```



```

        function_choice_realization(substitution_system, NAM);
    }
    cout << "Хотите ввести число ещё раз?" << '\n' << "Если да, нажмите Y, иначе любой другой
символ." << endl;
    cin >> answer;
    } while (answer == "Y");
}

bool check(string line)
{
    for (int i = 0; i < line.size(); i++)
    {
        if (i == 0 && line[i] == '0')
            return false;
        if (line[i] != '0' && line[i] != '1')
            return false;
    }
    return true;
}

void insert_substitution_system(vector<struct substitution>& ss)
{
    ss.push_back({ 1, "*0", "0*", false });
    ss.push_back({ 2, "*1", "1*", false });
    ss.push_back({ 3, "*", "!", false });
    ss.push_back({ 4, "000!", "!0", false });
    ss.push_back({ 5, "001!", "!1", false });
    ss.push_back({ 6, "010!", "!2", false });
    ss.push_back({ 7, "011!", "!3", false });
    ss.push_back({ 8, "100!", "!4", false });
    ss.push_back({ 9, "101!", "!5", false });
    ss.push_back({ 10, "110!", "!6", false });
    ss.push_back({ 11, "111!", "!7", false });
    ss.push_back({ 12, "10!", "!2", false });
    ss.push_back({ 13, "11!", "!3", false });
    ss.push_back({ 14, "1!", "!1", false });
    ss.push_back({ 15, "!", "", true });
    ss.push_back({ 16, " ", "*", false });
}

void function_choice_realization(vector<struct substitution>& ss, struct n_a_m& nam)
{
    bool flag1 = true;
    for (int i = 0; (i < ss.size()) && flag1; i++)
    {
        if (nam.line.find(ss[i].start) != -1)
        {
            flag1 = false;
            bool flag2 = true;
            int count;
            nam.stop = ss[i].stop;
            for (int j = 0; (j < nam.line.size()) && flag2; j++)
            {
                count = 0;
                for (int k = 0; k < ss[i].start.size(); k++)
                {
                    if (nam.line[j + k] == ss[i].start[k])
                        count++;
                }
                if (count == ss[i].start.size())
                {
                    flag2 = false;
                }
            }
        }
    }
}

```

```

        nam.position = j;
    }
}
cout << "(" << ss[i].number << ")" ";
for (int k = 0; k < nam.position; k++)
    cout << nam.line[k];
SetConsoleTextAttribute(hConsole, 12);
for (int k = nam.position; k < nam.position + count; k++)
    cout << nam.line[k];
SetConsoleTextAttribute(hConsole, 7);
for (int k = nam.position + count; k < nam.line.size(); k++)
    cout << nam.line[k];
nam.line.replace(nam.position, ss[i].start.size(), ss[i].finish);
cout << " -> ";
for (int k = 0; k < nam.line.size(); k++)
    cout << nam.line[k];
cout << endl;
if (ss[i].stop)
    cout << "Результат: " << nam.line << endl;
}
}

```