Индивидуальное задание №4

Проектирование и реализация нормального алгоритма Маркова

Патычкина Елизавета Вадимовна, КТбо1-10

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.

1. **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.

1. **Используемый алфавит ленты**

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

A.вх. = {0, 1} – входной алфавит нормального алгоритма Маркова

A.вых. = {0, 1, 2, 3, 4, 5, 6, 7} – выходной алфавит нормального алгоритма Маркова

A.всп. = {!, \*, ‘ ‘} – вспомогательный алфавит нормального алгоритма Маркова

1. **Система подстановок нормального алгоритма**
2. \*0 → 0\* - Двигаем метку в конец строки
3. \*1 → 1\* - Двигаем метку в конец строки
4. \* → ! - Меняем метку
5. 000! → !0 – С конца переводим из двоичной системы счисления в восьмеричную
6. 001! → !1 – С конца переводим из двоичной системы счисления в восьмеричную
7. 010! → !2 – С конца переводим из двоичной системы счисления в восьмеричную
8. 011! → !3 – С конца переводим из двоичной системы счисления в восьмеричную
9. 100! → !4 – С конца переводим из двоичной системы счисления в восьмеричную
10. 101! → !5 – С конца переводим из двоичной системы счисления в восьмеричную
11. 110! → !6 – С конца переводим из двоичной системы счисления в восьмеричную
12. 111! → !7 – С конца переводим из двоичной системы счисления в восьмеричную
13. 10! → !2 – Переводим из двоичной системы счисления в восьмеричную первое число без незначащих нулей
14. 11! → !3 – Переводим из двоичной системы счисления в восьмеричную первое число без незначащих нулей
15. 1! → !1 – Переводим из двоичной системы счисления в восьмеричную первое число без незначащих нулей
16. ! → . – Завершаем работу
17. → \* – Вводим маркер
18. **Набор тестов, охватывающих все режимы работы алгоритма и особые случаи**
19. 1000100

Входное слово: 1000100

Выходное слово: 104

1. 10011001

Входное слово: 10011001

Выходное слово: 231

1. 11101010

Входное слово: 11101010

Выходное слово: 352

1. 0111

Входное слово: 0111

Выходное слово: Неверный ввод

1. 12101

Входное слово: 12101

Выходное слово: Неверный ввод

1. 111111110

Входное слово: 111111110

Выходное слово: 776

1. **Скриншоты выполнения программы**
2. Тест 1

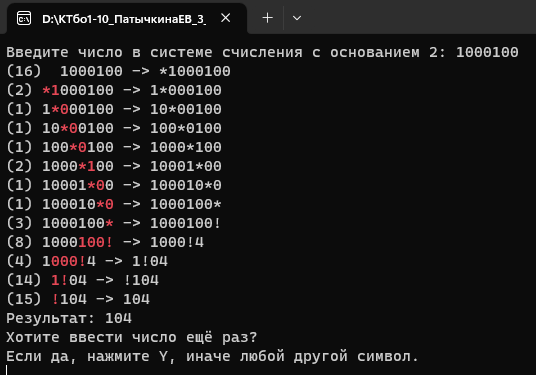


Рисунок 1 – Тест 1

1. Тест 2

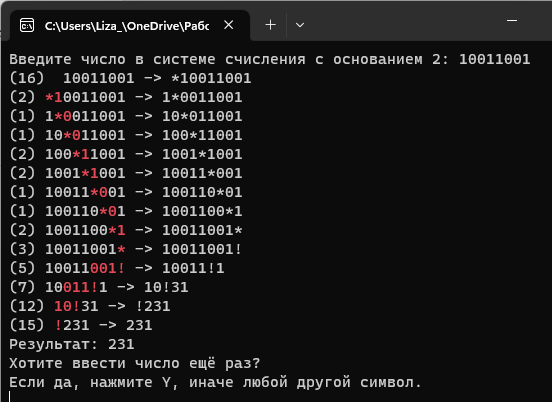


Рисунок 2 – Тест 2

1. Тест 3

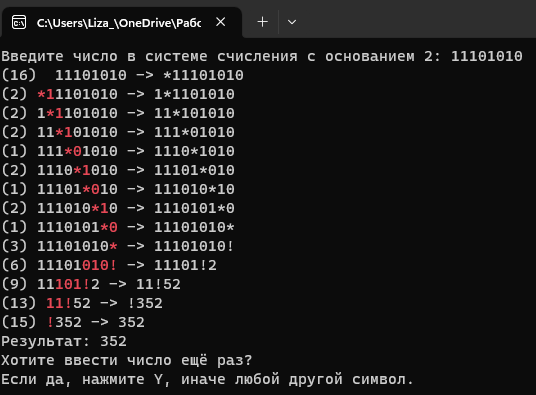


Рисунок 3 – Тест 3

1. Тест 4

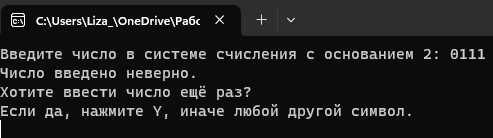


Рисунок 4 – Тест 4

1. Тест 5

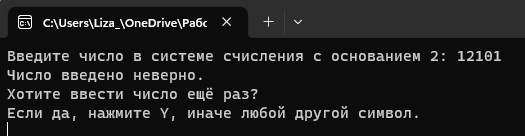


Рисунок 5 – Тест 5

1. Тест 6

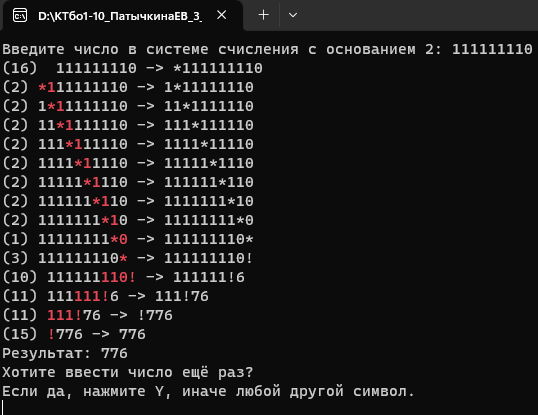


Рисунок 6 – Тест 6

1. **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.

1. **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

//КТбо1-10, 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;

}

}

}