**Лабораторная работа**

**Код Рида-Соломона**

**Цель работы** - познакомиться с алгоритмом кода Рида-Соломона для коррекции ошибок и освоить его реализацию на языке Python.

**Задачи работы**

* Понять, как код Рида-Соломона использует контрольные символы для обнаружения и исправления ошибок.
* Реализовать алгоритм кодирования и декодирования на Python.
* Смоделировать ошибку в передаче данных и исправить её с использованием кода Рида-Соломона.

**Теоретические сведения**

Код Рида-Соломона — это систематический блочный код, основанный на полиномиальных операциях в конечных полях. Он применяется для исправления ошибок в данных и используется в таких областях, как цифровая связь, хранение данных на оптических и магнитных носителях, QR-коды и многое другое.

**Основные этапы работы**

Кодирование — создание контрольных символов.

Декодирование — проверка данных на наличие ошибок и их исправление.

Кодирование осуществляется умножением полинома данных на генераторный полином и добавлением остатка от деления в виде контрольных символов.

**Задания к лабораторной работе**

**Задание 1: Разбор работы кода и анализ результата**

1. Проанализируйте код программы и опишите, как происходит кодирование и декодирование.
2. Выполните программу без внесения ошибок и запишите кодовое слово, полученное на этапе кодирования.
3. Объясните, почему при декодировании корректное получение данных не требует исправлений.

**Задание 2: Модификация алгоритма для разных позиций ошибок**

1. Измените позицию ошибки ERR\_POS на другое значение (например, 2-й и 5-й биты).
2. Объясните, как изменение позиции ошибки влияет на результаты декодирования.
3. Зафиксируйте позиции ошибок, которые программа может успешно исправить, и те, которые она не может исправить.

**Задание 3: Исследование ограничений корректирующих возможностей**

1. Измените значение контрольного полинома MAG и определите, как это изменение влияет на возможность обнаружения ошибок.
2. Внесите две ошибки в разные биты закодированного слова и попытайтесь выполнить декодирование.
3. Опишите, как программа реагирует на две ошибки, и обоснуйте, почему в некоторых случаях исправление невозможно.

**Задание 4: Реализация собственного метода поиска позиции ошибки**

1. Реализуйте свой метод для поиска позиции ошибки, не используя таблицу pow\_table. Например, напишите цикл, проверяющий остаток от деления на порождающий полином.
2. Сравните производительность и сложность обоих методов (с pow\_table и без).
3. Сделайте выводы о важности оптимизации алгоритмов для ускорения работы.

**Задание 5: Проверка кодирования на нескольких значениях данных**

1. Напишите программу, которая будет кодировать и декодировать несколько значений DATAIN (например, 0x23, 0xA5, 0x5C).
2. Для каждого значения DATAIN выполните кодирование и внесите ошибку в случайную позицию.
3. Проверьте, насколько успешно программа обнаруживает и исправляет ошибки для каждого значения данных.

**Задание 6: Расширение на множественные ошибки**

1. Измените код так, чтобы он пытался исправлять множественные ошибки (для этого потребуется модификация MAG и возможное добавление новых контрольных символов).
2. Исследуйте, как меняется производительность и количество исправляемых ошибок при увеличении числа контрольных символов.
3. Подготовьте отчет по результатам, сделав выводы о границах корректирующих возможностей кода Рида-Соломона в такой реализации.

*Простейший пример реализации кодера/декодера Рида-Соломона, работающего по обычной арифметике (т.е. с неоправданным расширением разрядной сетки), и исправляющего любые одиночные ошибки в одном 8-битном информационном слове (впрочем, программу легко адаптировать и под 16-битные информационные слова). Обратите внимание, что кодер реализуется чуть ли не на порядок проще декодера. В настоящем декодере Рида-Соломона, способном исправлять групповые ошибки, этот разрыв еще значительнее*

Язык С

#include <stdio.h>

// ширина входного информационного символа (бит)

#define SYM\_WIDE 8

// входные данные (один байт)

#define DATAIN 0x69

// номер бита, который будет разрушен сбоем

#define ERR\_POS 3

// неприводимый полином

#define MAG (1<<(SYM\_WIDE\*1) + 1<<(SYM\_WIDE\*0))

// -----------------------------------------------------

// определение позиции ошибки x по остатку k от деления

// кодового слова на полином k = 2^x, где "^" – возведение

// в степень; функция принимает k и возвращает x

// -----------------------------------------------------

int pow\_table[9] = {1, 2, 4, 8, 16, 32, 64, 128, 256};

int lockup(int x) {

int a;

for (a = 0; a < 9; a++)

if (pow\_table[a] == x) return a;

return -1;

}

int main() {

int i, g, c, e, k;

fprintf(stderr, "simplest Reed-Solomon encoder/decoder by Kris Kaspersky\n\n");

// входные данные (информационное слово)

i = DATAIN;

// неприводимый полином

g = MAG;

printf("i = %08x (DATAIN)\ng = %08x (POLYNOM)\n", i, g);

// КОДЕР РИДА-СОЛОМОНОНА (простейший, но все-таки кое-как работающий).

// Вычисляем кодовое слово, предназначенное для передачи

c = i \* g;

printf("c = %08x (CODEWORD)\n", c);

// конец КОДЕРА

// передаем с искажениями

e = c ^ (1 << ERR\_POS);

printf("e = %08x (RAW RECEIVED DATA+ERR)\n\n", e);

/\* ^^^^ искажаем один бит, имитируя ошибку передачи \*/

// ДЕКОДЕР РИДА-СОЛОМОНОНА

// проверяем на наличие ошибок передачи

// (фактически это простейший декодер Рида-Соломона)

if (e % g) {

// ошибки обнаружены, пытаемся исправить

printf("RS decoder says: (%x) error detected\n{\n", e % g);

// k = 2^x, где x - позиция сбойного бита

k = (e % g);

printf("\t0 to 1 err position: %x\n", lockup(k));

printf("\trestored codeword is: %x\n}\n", (e ^= k));

}

printf("RECEIVED DATA IS: %x\n", e / g);

// конец ДЕКОДЕРА

}

На Python

# ширина входного информационного символа (бит)

SYM\_WIDE = 8

# входные данные (один байт)

# Здесь мы используем байт данных 0x69 (в двоичной системе: 01101001)

DATAIN = 0x69

# номер бита, который будет разрушен сбоем

# Указываем бит, который будет изменён для имитации ошибки. В данном случае это 3-й бит.

ERR\_POS = 3

# неприводимый полином

# Это полином, который используется для кодирования. В данном случае полином представлен как сумма двух сдвигов: 1 << (SYM\_WIDE \* 1) и 1 << (SYM\_WIDE \* 0),

# что в итоге даёт нам полином 256 + 1 = 257.

MAG = (1 << (SYM\_WIDE \* 1)) + (1 << (SYM\_WIDE \* 0))

# Таблица степеней 2. Используется для поиска позиции ошибки.

# pow\_table хранит значения 2^x для x от 0 до 8.

pow\_table = [1, 2, 4, 8, 16, 32, 64, 128, 256]

# Функция поиска позиции ошибки по остатку k

# Она принимает число x и возвращает его позицию в таблице степеней 2.

# Если остаток k совпадает с каким-либо значением в pow\_table, возвращается его индекс (позиция бита ошибки).

def lockup(x):

for a in range(9):

if pow\_table[a] == x:

return a

return -1 # Если не найдено, возвращаем -1

# Основная функция программы

def main():

print("Simplest Reed-Solomon encoder/decoder \n")

# Входные данные (информационное слово)

i = DATAIN # Это наше информационное слово, которое мы будем кодировать

# Неприводимый полином

g = MAG # Порождающий полином, который используется для кодирования и декодирования

print(f"i = {i:08x} (DATAIN)\ng = {g:08x} (POLYNOM)")

# КОДЕР РИДА-СОЛОМОНА

# Кодирование: умножаем информационное слово i на порождающий полином g, получая кодовое слово c

c = i \* g

print(f"c = {c:08x} (CODEWORD)")

# конец КОДЕРА

# Передаем данные с искажениями

# Мы меняем один бит в кодовом слове для имитации ошибки в передаче

# XOR (^) меняет один бит, который мы указываем с помощью сдвига 1 << ERR\_POS

e = c ^ (1 << ERR\_POS)

print(f"e = {e:08x} (RAW RECEIVED DATA+ERR)\n")

# Это симуляция передачи данных с ошибкой в 3-м бите.

# ДЕКОДЕР РИДА-СОЛОМОНА

# Проверка на наличие ошибок в полученных данных

# Если остаток от деления (e % g) не равен 0, значит была ошибка

if e % g != 0:

# Ошибка обнаружена. Пытаемся найти и исправить её.

print(f"RS decoder says: ({e % g:x}) error detected\n{{")

# k = e % g, где k — это остаток от деления. Это число соответствует 2^x, где x — позиция сбойного бита.

k = e % g

# lockup(k) ищет позицию ошибки, соответствующую остатку k

print(f"\t0 to 1 err position: {lockup(k):x}")

# Исправляем кодовое слово, используя XOR для отмены ошибки

e ^= k

print(f"\trestored codeword is: {e:x}\n}}")

# Полученные данные после исправления ошибки

# Здесь мы делим исправленное кодовое слово e на порождающий полином g, чтобы получить исходные данные

print(f"RECEIVED DATA IS: {e // g:x}")

# Запуск основного блока программы

if \_\_name\_\_ == "\_\_main\_\_":

main()

***Пояснения***

DATAIN: Это исходные данные (информационное слово), которые мы хотим передать.

ERR\_POS: Этот параметр указывает на бит, который будет изменён для имитации ошибки при передаче данных.

MAG: Это порождающий полином, который используется для кодирования и декодирования. Он определяет, как данные будут кодироваться.

pow\_table: Таблица степеней двойки, используемая для поиска позиции ошибки по остатку деления.

lockup(x): Эта функция ищет, какой бит был искажён (где находится ошибка), используя таблицу степеней 2.

Основная функция:

- Сначала кодируется информационное слово путём умножения на порождающий полином.

- Затем происходит передача с ошибкой (один бит изменяется).

- После этого код пытается обнаружить и исправить ошибку.

Кодирование: Умножение исходных данных на полином даёт закодированное слово.

Декодирование: Ошибка обнаруживается путём проверки остатка от деления. Если остаток не равен нулю, значит, произошла ошибка, которую мы можем найти и исправить.

Результат работы простейшего кодера/декодера Рида-Соломона. Обратите внимание: искаженный бит удалось успешно исправить, однако для этого к исходному информационному слову пришлось добавить не два, а целых три бита (если вы возьмете в качестве входного слова максимально допустимое 8-битное значение 0xFF, то кодовое слово будет равно 0x1FE00, а так как 210 = 10000, то свободных разрядов уже не хватает и приходится увеличивать разрядную сетку до 211, в то время как младшие биты кодового слова фактически остаются незадействованными и “правильный” кодер должен их “закольцевать”, грубо говоря замкнув обрабатываемые разряды на манер кольца.

i = 00000069 (DATAIN)

g = 00000200 (POLYNOM)

c = 0000d200 (CODEWORD)

e = 0000d208 (RAW RECEIVED DATA+ERR)

RS decoder says: (8) error detected

{

0 to 1 err position: 3

restored codeword is: d200

}

RECEIVED DATA IS: 69