МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Кафедра ИИТ

ОТЧЁТ
По лабораторной работе №2
«Избыточное кодирование данных в информационных системах. Итеративные коды»

Выполнил:
Студент группы ИИ-22
Кузьмич В.Н.
Проверила:
Хацкевич А.С.

Брест 2024

Цель работы: приобретение практических навыков кодирования/декодирования двоичных данных при использовании итеративных кодов.

Задание.

1. Закрепить теоретические знания по использованию итеративных кодов для повышения надежности передачи и хранения в памяти компьютера двоичных данных.
2. Разработать приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

Ход работы

Вариант	Длина слова, бит	k_1	k_2	Z	Кол-во групп паритетов
2	20	4	5	-	2,3
		2	10	-	2,3
		2	5	2	2,3,4,5
		2	2	5	2,3,4,5

Код программы:

```
import numpy as np

def add_errors(binary_word, num_errors):
    """Добавляет случайные ошибки в двоичное слово."""
    error_indices = np.random.choice(len(binary_word), size=num_errors, replace=False)
    binary_word_with_errors = binary_word.copy()
    binary_word_with_errors[error_indices] = np.bitwise_xor(binary_word_with_errors[error_indices], 1)
    return binary_word_with_errors, error_indices

class IterativeCode:
    def __init__(self, length, rows, cols, n_parities):
        self.length = length
        self.rows = rows
        self.cols = cols
        self.n_parities = n_parities
        self.word = self.generate_word()
        self.matrix = self.word2matrix()
        self.parities = self.calculate_parities()

    def generate_word(self):
        """Генерирует случайное двоичное слово длины length."""
        return np.random.randint(2, size=self.length)

    def word2matrix(self):
        """Преобразует двоичное слово в матрицу с заданными размерами."""
        return self.word.reshape((self.rows, self.cols))

    def calculate_parities(self):
        """Вычисляет паритетные биты для указанных направлений."""
        parities = {}
        if self.n_parities >= 2:
            parities['row'] = np.sum(self.matrix, axis=1) % 2
            parities['col'] = np.sum(self.matrix, axis=0) % 2
        if self.n_parities >= 3:
            parities['diag_down'] = self.calculate_diagonal_parity_down()
        if self.n_parities >= 4:
            parities['diag_up'] = self.calculate_diagonal_parity_up()
        return parities

    def calculate_diagonal_parity_up(self):
        """Вычисляет паритетные биты по восходящим диагоналям."""
        rows, cols = self.matrix.shape
        parities = []
        for offset in range(-(rows - 1), cols):
            diag = np.diagonal(self.matrix, offset=offset)
            parity = np.sum(diag) % 2
            parities.append(parity)
        return np.array(parities)

    def calculate_diagonal_parity_down(self):
        """Вычисляет паритетные биты по нисходящим диагоналям."""
        flipped_matrix = np.fliplr(self.matrix)
```

```
        rows, cols = flipped_matrix.shape
        parities = []
        for offset in range(-(rows - 1), cols):
            diag = np.diagonal(flipped_matrix, offset=offset)
            parity = np.sum(diag) % 2
            parities.append(parity)
        return np.array(parities)[::-1]

    def get_indices(self, direction, index):
        """Возвращает индексы элементов, входящих в группу паритета для строки, столбца или диагонали."""
        if direction == 'row':
            return self.get_row_indices(index)
        elif direction == 'col':
            return self.get_col_indices(index)
        elif direction == 'diag_down':
            return self.get_diagonal_indices_down(index)
        elif direction == 'diag_up':
            return self.get_diagonal_indices_up(index)

    def get_row_indices(self, row_index):
        """Возвращает индексы для строки."""
        return [(row_index, col_idx) for col_idx in range(self.matrix.shape[1])]

    def get_col_indices(self, col_index):
        """Возвращает индексы для столбца."""
        return [(row_idx, col_index) for row_idx in range(self.matrix.shape[0])]

    def get_diagonal_indices_up(self, parity_index):
        """Возвращает индексы для восходящей диагонали."""
        rows, cols = self.matrix.shape
        offset = parity_index - (rows - 1)
        diag = np.diagonal(self.matrix, offset=offset)
        if offset >= 0:
            return [(i, i + offset) for i in range(len(diag))]
        else:
            return [(i - offset, i) for i in range(len(diag))]

    def get_diagonal_indices_down(self, parity_index):
        """Возвращает индексы для нисходящей диагонали."""
        indices = self.get_diagonal_indices_up(parity_index)
        return [(self.rows - 1 - index[0], index[1]) for index in indices]

    def __str__(self):
        return f"Слово: {self.word}\n" + \
               f"Матрица:\n {self.matrix}\n" + \
               f"Паритеты строк: {self.parities['row']}\n" + \
               f"Паритеты столбцов: {self.parities.get('col')}\n" + \
               f"Паритеты диагонали (вниз): {self.parities.get('diag_down')}\n" + \
               f"Паритеты диагонали (вверх): {self.parities.get('diag_up')}\n"

class IterativeCodeSend(IterativeCode):
    def combine_parities_and_word(self):
        """Объединяет исходное слово и вычисленные паритетные биты."""
        parities_array = [self.word]
        for key in list(self.parities.keys()):
            parities_array.append(self.parities[key])
        return np.concatenate(parities_array)

class IterativeCodeReceive(IterativeCode):
    def __init__(self, length, rows, cols, n_parities, word):
        super().__init__(length, rows, cols, n_parities)
        self.unpack(word)
        self.matrix = self.word2matrix()
        self.parities = self.calculate_parities()
        self.errors = self.find_errors()

    def unpack(self, word):
        """Извлекает кодовое слово и паритетные биты из принятого слова."""
        splits = [
            self.length,
            self.length + self.rows,
            self.length + self.rows + self.cols,
            self.length + 2 * (self.rows + self.cols) - 1]
        self.word = word[:splits[0]]
        self.current_parities = {}
        if self.n_parities >= 2:
            self.current_parities['row'] = word[splits[0]:splits[1]]
            self.current_parities['col'] = word[splits[1]:splits[2]]
        if self.n_parities >= 3:
            self.current_parities['diag_down'] = word[splits[2]:splits[3]]
        if self.n_parities >= 4:
            self.current_parities['diag_up'] = word[splits[3]:]

    def find_errors(self):
        """Находит ошибки, сравнивая вычисленные и полученные паритетные биты."""
        errors = {}
```

```
        for key in list(self.parities.keys()):
            errors_in_parities = (np.where(self.current_parities[key] != self.parities[key])[0]).tolist()
            errors[key] = set()
            for error_index in errors_in_parities:
                errors[key].update(self.get_indices(key, error_index))
        if len(errors.keys()) < self.n_parities:
            return set()
        return set.intersection(*errors.values())

    def fix_errors(self):
        """Исправляет ошибки в принятом кодовом слове."""
        if not self.errors:
            return self.word
        matrix = self.matrix.copy()
        for x, y in self.errors:
            matrix[x][y] ^= 1
        return matrix.flatten()

    def __str__(self):
        return super().__str__() + \
               f"Найденные ошибки: {self.errors}\n" + \
               f"Исправленное слово: {self.fix_errors()}"

if __name__ == "__main__":
    length = 20
    rows, cols = 4, 5
    num_parities = 4
    num_errors = 2

    code2send = IterativeCodeSend(length, rows, cols, num_parities)
    print(code2send)

    word2send = code2send.combine_parities_and_word()
    print("Слово с паритетами:", word2send)

    word2send_with_errors, error_positions = add_errors(word2send, num_errors)
    print("Слово с ошибками:", word2send_with_errors)
    print("Позиции ошибок:", error_positions)

    code2receive = IterativeCodeReceive(length, rows, cols, num_parities, word2send_with_errors)
    print(code2receive)
```

Результат работы:

```
Слово: [1 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1]
Матрица:
 [[1 1 1 1 1]
 [0 0 1 1 1]
 [0 0 0 1 1]
 [1 1 0 1 1]]
Паритеты строк: [1 1 0 0]
Паритеты столбцов: [0 0 0 0 0]
Паритеты диагонали (вниз): [1 1 1 1 1 0 0 1]
Паритеты диагонали (вверх): [1 1 0 0 0 1 0 1]

Слово с паритетами: [1 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1
 1 1 0 0 0 1 0 1]
Слово с ошибками: [1 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1
 1 1 0 0 0 1 0 1]
Позиции ошибок: [30 32]
Слово: [1 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1]
Матрица:
 [[1 1 1 1 1]
 [0 0 1 1 1]
 [0 0 0 1 1]
 [1 1 0 1 1]]
Паритеты строк: [1 1 0 0]
Паритеты столбцов: [0 0 0 0 0]
Паритеты диагонали (вниз): [1 1 1 1 1 0 0 1]
Паритеты диагонали (вверх): [1 1 0 0 0 1 0 1]
Найденные ошибки: set()
Исправленное слово: [1 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1]
```

Вывод: приобрёл практические навыки кодирования/декодирования двоичных данных при использовании итеративных кодов.