

Министерство образования Республики Беларусь  
Учреждение образования  
“Брестский государственный технический университет” Кафедра  
интеллектуально-информационных технологий

Лабораторная работа №2  
“Избыточное кодирование данных в информационных системах.  
Итеративные коды”

Выполнил:  
студент 4 курса  
группы ИИ-22  
Нестерчук Д.Н.  
Проверила:  
Хацкевич А. С.

Брест 2024

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

**Ход работы:**

### Задание

Разработать собственное приложение, которое позволяет выполнять следующие операции:

- 1) вписывать произвольное двоичное представление информационного слова  $X_k$  (кодируемой информации) длиной  $k$  битов в двумерную матрицу размерностью в соответствии с вариантом;
- 2) вычислить проверочные биты (биты паритетов): а) по двум; б) по трем; в) по четырем направлениям (группам паритетов);
- 3) формировать кодовое слово  $X_n$  присоединением избыточных символов к информационному слову;
- 4) генерировать ошибку произвольной кратности ( $i, i > 0$ ), распределенную случайным образом среди символов слова  $X_n$ , в результате чего формируется кодовое слово  $Y_n$ ;
- 5) определять местоположение ошибочных символов итеративным кодом в слове  $Y_n$  в соответствии с используемыми группами паритетов и исправлять ошибочные символы (результат исправления - слово  $Y_n'$ );
- 6) выполнять анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов  $X_n$  и  $Y_n'$ ; результат анализа может быть представлен в виде отношения общего числа сгенерированных кодовых слов с ошибками определенной одинаковой кратности (с одной ошибкой, с двумя ошибками и т. д.) к числу кодовых слов, содержащих ошибки этой кратности, которые правильно обнаружены и которые правильно скорректированы.

### Вариант 2

Варианты задания

Вариант	Длина информационного слова (бит), $k$	$k_1$	$k_2$	$z$	Количество групп паритетов
1	16	4	4	–	2; 3
		8	2	–	2; 3
		4	2	2	2; 3; 4; 5
		2	4	2	2; 3; 4; 5
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 без применения:**

```
import random
import numpy as np

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.word_to_matrix()
        self.parities = self.calculate_parities()

    # Генерация случайного двоичного слова
    def generate_word(self):
        return [random.randint(0, 1) for _ in
range(self.length)]

    # Преобразование слова в матрицу
    def word_to_matrix(self):
        matrix = np.zeros((self.rows, self.cols), dtype=int)
        for i in range(self.length):
            matrix[i // self.cols][i % self.cols] = self.word[i]
        return matrix

    # Вычисление паритетов (строки, столбцы, диагонали)
    def calculate_parities(self):
        parities = {}
        if self.n_parities >= 2:
            parities['row'] = self.calculate_row_parity()
            parities['col'] = self.calculate_col_parity()
        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

    # Вычисление паритетов по строкам
    def calculate_row_parity(self):
        return [sum(row) % 2 for row in self.matrix]

    # Вычисление паритетов по столбцам
    def calculate_col_parity(self):
```

```
        return [sum(self.matrix[:, col]) % 2 for col in
range(self.cols)]
```

```
# Вычисление диагональных паритетов вверх
def calculate_diagonal_parity_up(self):
    parities = []
    for offset in range(-(self.rows - 1), self.cols):
        parity = 0
        for i in range(max(0, -offset), min(self.rows,
self.cols - offset)):
            parity ^= self.matrix[i][i + offset]
        parities.append(parity)
    return parities
```

```
# Вычисление диагональных паритетов вниз
def calculate_diagonal_parity_down(self):
    parities = []
    for offset in range(-(self.rows - 1), self.cols):
        parity = 0
        for i in range(max(0, -offset), min(self.rows,
self.cols - offset)):
            parity ^= self.matrix[self.rows - 1 - i][i +
offset]
        parities.append(parity)
    return parities
```

```
def __str__(self):
    # Преобразуем паритеты в стандартные числа (int), чтобы
    избежать вывода pr.int64
    return f"Слово: {self.word}\n" \
           f"Матрица: \n{self.matrix}\n" \
           f"Паритеты строк:
{self._convert_to_int(self.parities.get('row', []))}\n" \
           f"Паритеты столбцов:
{self._convert_to_int(self.parities.get('col', []))}\n" \
           f"Паритеты диагоналей вниз:
{self._convert_to_int(self.parities.get('diag_down', []))}\n" \
           f"Паритеты диагоналей вверх:
{self._convert_to_int(self.parities.get('diag_up', []))}\n"
```

```
def _convert_to_int(self, parity_list):
    # Преобразует все элементы в список обычных int
    return [int(p) for p in parity_list]
```

[illegible]

```
return combined
```

```
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.word_to_matrix()
        self.parities = self.calculate_parities()
        self.errors = self.find_errors()

    def unpack(self, word):
        self.word = word[:self.length]
        idx = self.length
        if self.n_parities >= 2:
            self.current_parities = word[idx: idx + self.rows]
            idx += self.rows
        if self.n_parities >= 3:
            self.current_parities += word[idx: idx + self.rows +
self.cols - 1]
            idx += self.rows + self.cols - 1
        if self.n_parities >= 4:
            self.current_parities += word[idx: idx + self.rows +
self.cols - 1]

    def find_errors(self):
        errors = []
        for key, parity_array in self.parities.items():
            for i, parity in enumerate(parity_array):
                if parity != self.current_parities[i]:
                    errors.extend(self.get_indices(key, i))
        return errors

    def get_indices(self, key, index):
        if key == "row":
            return self.get_row_indices(index)
        elif key == "col":
            return self.get_col_indices(index)
        elif key == "diag_down":
            return self.get_diagonal_indices_down(index)
        elif key == "diag_up":
            return self.get_diagonal_indices_up(index)
        return []

    def get_row_indices(self, row_index):
        return [(row_index, col) for col in range(self.cols)]

    def get_col_indices(self, col_index):
        return [(row, col_index) for row in range(self.rows)]

    def get_diagonal_indices_up(self, parity_index):
        indices = []
        offset = parity_index - (self.rows - 1)
```

```

        for i in range(max(0, -offset), min(self.rows, self.cols
- offset)):
            indices.append((i, i + offset))
        return indices

    def get_diagonal_indices_down(self, parity_index):
        indices = []
        offset = parity_index - (self.rows - 1)
        for i in range(max(0, -offset), min(self.rows, self.cols
- offset)):
            indices.append((self.rows - 1 - i, i + offset))
        return indices

    def fix_errors(self):
        for error in self.errors:
            row, col = error
            self.matrix[row, col] ^= 1
        return self.matrix.flatten()

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

```

```

class ErrorGenerator:
    @staticmethod
    def add_errors(binary_word, num_errors):
        error_indices = set()
        while len(error_indices) < num_errors:
            error_indices.add(random.randint(0, len(binary_word)
- 1))
        binary_word_with_errors = binary_word.copy()
        for idx in error_indices:
            binary_word_with_errors[idx] ^= 1
        return binary_word_with_errors

```

```

# Пример использования
length = 16
rows = 4
cols = 4
n_paritys = 4

```

```

# Создание и отправка
iterative_send = IterativeCodeSend(length, rows, cols,
n_paritys)
combined_word = iterative_send.combine_paritys_and_word()
print("Отправленное слово и паритеты:", combined_word)

```

```

# Допустим, ошибка была внесена в закодированное слово
corrupted_word = ErrorGenerator.add_errors(combined_word, 1)

```

```

# Прием и обработка

```

```
iterative_receive = IterativeCodeReceive(length, rows, cols,
n_parities, corrupted_word)
print(iterative_receive)
```

```
# Исправление ошибок
```

```
corrected_word = iterative_receive.fix_errors()
print("Исправленное слово:", corrected_word)
```

### Вывод программы:

```
C:\Users\Xiaomi\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Xiaomi\Desktop\prjs\SMZKS-
Закодированное сообщение: [1 1 0 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1]
Ошибка в группе паритета 1
Ошибка в группе паритета 2
Ошибка в группе паритета 3
Ошибка в группе паритета 4
Обнаружены ошибки в принятых данных.
Декодированное сообщение: [1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 0 1 0 1]
```