

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  
Клебанович В. 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	Длина информационного слова (бит), $k$	$k_1$	$k_2$	$z$	Количество групп паритетов
3	24	4	6	-	2;3
		3	8	-	2;3
		3	2	4	2;3;4;5
		6	2	2	2;3;4;5

Код программы без применения z:

```
import numpy as np

def generate_word(length, word=None):
    if word is not None:
        return np.array([int(bit) for bit in word])
    return np.random.randint(2, size=length)

def word_to_matrix(word, rows, cols):
    return word.reshape((rows, cols))

def calculate_parities(matrix, n_parities):
    parities = {}
    if n_parities >= 2:
        parities['row'] = np.sum(matrix, axis=1) % 2
        print(f"Паритеты по строкам: {parities['row']}")

        parities['col'] = np.sum(matrix, axis=0) % 2
        print(f"Паритеты по столбцам: {parities['col']}")

    if n_parities >= 3:
        parities['diag_down'] = calculate_diagonal_parity_down(matrix)
        print(f"Паритеты диагонали вниз: {parities['diag_down']}")

    if n_parities >= 4:
        parities['diag_up'] = calculate_diagonal_parity_up(matrix)
        print(f"Паритеты диагонали вверх: {parities['diag_up']}")

    return parities

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

def calculate_diagonal_parity_down(matrix):
    flipped_matrix = np.fliplr(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 calculate_parity_of_paritys(parities):
    total_parity = 0
    for parity in parities.values():
        total_parity ^= np.sum(parity) % 2 # XOR всех паритетов
    return total_parity

def generate_encoded_word(length, rows, cols, n_paritys, word=None):
    word_array = generate_word(length, word)
    print(f"Первоначальное слово: {word_array}")

    matrix = word_to_matrix(word_array, rows, cols)
    print(f"Матрица:\n{matrix}")

    parities = calculate_paritys(matrix, n_paritys)

    # Объединяем все паритеты в одно кодовое слово
    encoded_word = list(word_array)

    if 'row' in parities:
        encoded_word.extend(parities['row'])
    if 'col' in parities:
        encoded_word.extend(parities['col'])
    if 'diag_down' in parities:
        encoded_word.extend(parities['diag_down'])
    if 'diag_up' in parities:
        encoded_word.extend(parities['diag_up'])

    # Добавляем паритет паритетов
    parity_of_paritys = calculate_parity_of_paritys(parities)
    encoded_word.append(parity_of_paritys)

    print(f"Закодированное слово: {encoded_word}\n")
    return np.array(encoded_word), parities

def introduce_error(encoded_word, position):
    """Вводит ошибку в закодированное слово на заданной позиции."""
    error_word = encoded_word.copy()
    error_word[position] ^= 1 # Изменяем бит (XOR с 1)
    return error_word

def correct_error(encoded_word, rows, cols, n_paritys):
    """Исправляет ошибку в закодированном слове."""
    # Извлекаем информацию из закодированного слова
    data_length = rows * cols
    parities_length = (rows + cols) + (rows + cols - 1) + (rows + cols - 1) + 1
    expected_length = data_length + parities_length
    assert len(encoded_word) == expected_length, "Длина закодированного слова неверна!"

```

```

# Извлекаем данные и паритеты
data = encoded_word[:data_length]
row_parities = encoded_word[data_length:data_length + rows]
col_parities = encoded_word[data_length + rows:data_length + rows + cols]
diag_down_parities = encoded_word[data_length + rows + cols:data_length + rows
+ cols + (rows + cols - 1)]
diag_up_parities = encoded_word[data_length + rows + cols + (rows + cols - 1):-1]
total_parity = encoded_word[-1]

# Восстанавливаем матрицу
matrix = word_to_matrix(data, rows, cols)
print(f"\nВосстановленная матрица:\n{matrix}")

# Рассчитываем актуальные паритеты
calculated_parities = calculate_parities(matrix, n_parities)

# Проверка паритетов
row_errors = row_parities != calculated_parities['row']
col_errors = col_parities != calculated_parities['col']
diag_down_errors = diag_down_parities != calculated_parities['diag_down']
diag_up_errors = diag_up_parities != calculated_parities['diag_up']

error_row = np.where(row_errors)[0]
error_col = np.where(col_errors)[0]
if len(error_row) == 1 and len(error_col) == 1:
    error_position = error_row[0] * cols + error_col[0]
    print(f"Ошибка обнаружена на позиции {error_position}. Исправление...")
    matrix[error_row[0], error_col[0]] ^= 1
else:
    print("Ошибка не может быть исправлена или не обнаружена.")

# Возвращаем исправленное закодированное слово
corrected_word = matrix.flatten()
corrected_encoded_word = np.concatenate((corrected_word,
calculate_parities(matrix, n_parities)['row'],
                                calculate_parities(matrix, n_parities)['col'],
                                calculate_diagonal_parity_down(matrix),
                                calculate_diagonal_parity_up(matrix),
                                [calculate_parity_of_parities(calculate_parities(matrix,
n_parities))]))

return corrected_encoded_word

def main():
    length = 24
    rows, cols = 3, 8
    num_parities = 4

    fixed_word = "100110001110011000111001"

```

```
encoded_word, parities = generate_encoded_word(length, rows, cols, num_parities,
fixed_word)
```

```
print(f"\nСлово: {fixed_word}\n")
```

```
error_position = int(input("Введите позицию, где хотите ввести ошибку (0-{}) :
".format(len(encoded_word) - 1)))
corrupted_word = introduce_error(encoded_word, error_position)
```

```
print(f"\nЗакодированное слово с ошибкой: {corrupted_word}\n")
```

```
corrected_word = correct_error(corrupted_word, rows, cols, num_parities)
```

```
print(f"\nИсправленное закодированное слово: {corrected_word}\n")
```

```
if __name__ == "__main__":
    main()
```

## Вывод программы:

```
Первоначальное слово: [1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 1]
Матрица:
[[1 0 0 1 1 0 0 0]
 [1 1 1 0 0 1 1 0]
 [0 0 1 1 1 0 0 1]]
Паритеты по строкам: [1 1 0]
Паритеты по столбцам: [0 1 0 0 0 1 1 1]
Паритеты диагонали вниз: [1 1 1 0 0 1 0 1 0 1]
Паритеты диагонали вверх: [0 1 1 0 1 1 0 0 0 0]
Закодированное слово: [1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0]

Слово: 100110001110011000111001

Введите позицию, где хотите ввести ошибку (0-55) : 1

Закодированное слово с ошибкой: [1 1 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1
1 0 0 1 0 1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0]
```

```
Восстановленная матрица:
[[1 1 0 1 1 0 0 0]
 [1 1 1 0 0 1 1 0]
 [0 0 1 1 1 0 0 1]]
Паритеты по строкам: [0 1 0]
Паритеты по столбцам: [0 0 0 0 0 1 1 1]
Паритеты диагонали вниз: [1 0 1 0 0 1 0 1 0 1]
Паритеты диагонали вверх: [0 1 1 1 1 1 0 0 0 0]
Ошибка обнаружена на позиции 1. Исправление...
Паритеты по строкам: [1 1 0]
Паритеты по столбцам: [0 1 0 0 0 1 1 1]
Паритеты диагонали вниз: [1 1 1 0 0 1 0 1 0 1]
Паритеты диагонали вверх: [0 1 1 0 1 1 0 0 0 0]
Паритеты по строкам: [1 1 0]
Паритеты по столбцам: [0 1 0 0 0 1 1 1]
Паритеты диагонали вниз: [1 1 1 0 0 1 0 1 0 1]
Паритеты диагонали вверх: [0 1 1 0 1 1 0 0 0 0]
Паритеты по строкам: [1 1 0]
Паритеты по столбцам: [0 1 0 0 0 1 1 1]
Паритеты диагонали вниз: [1 1 1 0 0 1 0 1 0 1]
Паритеты диагонали вверх: [0 1 1 0 1 1 0 0 0 0]

Исправленное закодированное слово: [1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1
1 0 0 1 0 1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0]
```

## Код программы с применением z:

```
import numpy as np

message = "100110001110011000111001"

def split_message_into_matrices(message, matrix_size=(3, 2)):
    matrices = []
    step = matrix_size[0] * matrix_size[1]
    for i in range(0, len(message), step):
        part = message[i:i+step]
        if len(part) < step:
            part += '0' * (step - len(part))
        matrix = create_matrix(part)
        matrices.append(matrix)
    return matrices

def create_matrix(binary_str):
    matrix = []
    for i in range(0, len(binary_str), 2):
        if i + 1 < len(binary_str):
            matrix.append([int(binary_str[i]), int(binary_str[i + 1])])
    return np.array(matrix)

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

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

def calculate_diagonal_parity_down(matrix):
    flipped_matrix = np.fliplr(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 calculate_parity_of_parities(parities_list):
    total_parity = 0
    for parities in parities_list:
        for parity in parities.values():
            total_parity ^= np.sum(parity) % 2
    return total_parity

def calculate_parities_for_matrices(matrices, n_parities):
    parities_list = []
    for matrix in matrices:
        parities = calculate_parities(matrix, n_parities)
        parities_list.append(parities)
    return parities_list

def calculate_iterative_code(matrices):
    if len(matrices) < 2:
        return matrices[0]
    iterative_code = matrices[0]
    for matrix in matrices[1:]:
        iterative_code = (iterative_code + matrix) % 2
    return iterative_code

def parities_to_string(parities):
    parity_str = ""
    if 'row' in parities:
        parity_str += ".join(map(str, parities['row']))"
    if 'col' in parities:
        parity_str += ".join(map(str, parities['col']))"
    if 'diag_down' in parities:
        parity_str += ".join(map(str, parities['diag_down']))"
    if 'diag_up' in parities:
        parity_str += ".join(map(str, parities['diag_up']))"
    return parity_str

def form_codeword(message, parities_list, parity_of_parities, iterative_code):
    codeword = message
    for parities in parities_list:
        codeword += parities_to_string(parities)
    codeword += str(parity_of_parities)
    iterative_code_str = ".join(map(str, iterative_code.flatten()))"
    codeword += iterative_code_str
    return codeword

matrix_size = (3, 2)
matrices = split_message_into_matrices(message, matrix_size)

```



```

for idx, matrix in enumerate(matrices):
    print(f"Матрица {idx + 1}:\n", matrix)

n_parities = 4
parities_list = calculate_parities_for_matrices(matrices, n_parities)

for idx, parities in enumerate(parities_list):
    print(f"\nПаритеты для Матрицы {idx + 1}:")
    print(f"Паритеты строк: {parities.get('row')}")
    print(f"Паритеты столбцов: {parities.get('col')}")
    print(f"Паритеты диагонали (вправо): {parities.get('diag_down')}")
    print(f"Паритеты диагонали (влево): {parities.get('diag_up')}")

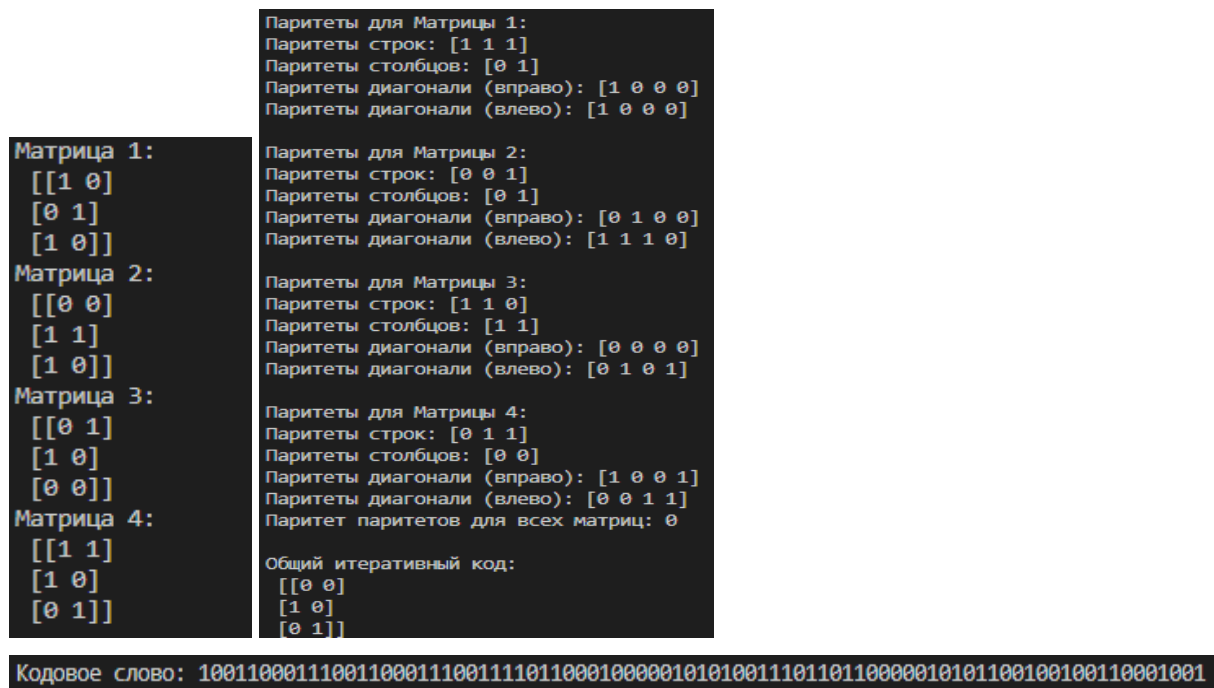
parity_of_parities = calculate_parity_of_parities(parities_list)
print(f"Паритет паритетов для всех матриц: {parity_of_parities}")

iterative_code = calculate_iterative_code(matrices)
print("\nОбщий итеративный код:\n", iterative_code)

codeword = form_codeword(message, parities_list, parity_of_parities, iterative_code)
print("\nКодовое слово:", codeword)
print("\nДлина кодового слова:", len(codeword))

```

## Вывод программы:



Матрица 1:

```
[[1 0]
 [0 1]
 [1 0]]
```

Матрица 2:

```
[[0 0]
 [1 1]
 [1 0]]
```

Матрица 3:

```
[[0 1]
 [1 0]
 [0 0]]
```

Матрица 4:

```
[[1 1]
 [1 0]
 [0 1]]
```

Паритеты для Матрицы 1:

```
Паритеты строк: [1 1 1]
Паритеты столбцов: [0 1]
Паритеты диагонали (вправо): [1 0 0 0]
Паритеты диагонали (влево): [1 0 0 0]
```

Паритеты для Матрицы 2:

```
Паритеты строк: [0 0 1]
Паритеты столбцов: [0 1]
Паритеты диагонали (вправо): [0 1 0 0]
Паритеты диагонали (влево): [1 1 1 0]
```

Паритеты для Матрицы 3:

```
Паритеты строк: [1 1 0]
Паритеты столбцов: [1 1]
Паритеты диагонали (вправо): [0 0 0 0]
Паритеты диагонали (влево): [0 1 0 1]
```

Паритеты для Матрицы 4:

```
Паритеты строк: [0 1 1]
Паритеты столбцов: [0 0]
Паритеты диагонали (вправо): [1 0 0 1]
Паритеты диагонали (влево): [0 0 1 1]
Паритет паритетов для всех матриц: 0
```

Общий итеративный код:

```
[[0 0]
 [1 0]
 [0 1]]
```

Кодовое слово: 10011000111001100011100111101100010000010101001110110110000010101100100100110001001

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