МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет
по дисциплине
«Современные методы защиты компьютерных систем»
по лабораторной работе № 2
«Избыточное кодирование данных в информационных системах. Итеративные коды»

Выполнила:
студентка 4
курса группы
ИИ-22
Сокол С.М.
Проверила:
Хацкевич А.С.

Брест 2024

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

**Постановка задачи:** Разработать приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.

| | | | | | |
|---|---|---|---|---|---|
| 4 | 32 | 4<br>2<br>8<br>4 | 8<br>16<br>2<br>4 | -<br>-<br>2<br>2 | 2; 3<br>2; 3<br>2; 3; 4; 5<br>2; 3; 4; 5 |

Ход работы: Была написана программа на языке Python, для реализации необходимых требований

Код:

```
2_1 import numpy as np
  def add_errors(binary_word, num_errors):
      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):
          return np.random.randint(2, size=self.length)

      def word2matrix(self):
          return self.word.reshape((self.rows, self.cols))  #type: ignore

      def calculate_parities(self):
          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):
          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):
        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, str, index):
        match str:
            case 'row':
                return self.get_row_indices(index)
            case 'col':
                return self.get_col_indices(index)
            case 'diag_down':
                return self.get_diagonal_indices_down(index)
            case 'diag_up':
                return self.get_diagonal_indices_up(index)

    def get_row_indices(self, row_index):
        return [(row_index, col_idx) for col_idx in range(self.matrix.shape[1])]

    def get_col_indices(self, col_index):
        return [(row_idx, col_index) for row_idx in range(self.matrix.shape[0])]

    def get_diagonal_indices_up(self, parity_index):
        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):
        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):
        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):
        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 calculate_parity(self, key):
        return self.parities.get(key)

    def find_errors(self):
        errors = {}
        for key in list(self.parities.keys()):
                            errors_in_parities  =  (np.where(self.current_parities[key]  !=
self.parities[key])[0]).tolist()
            for error_index in errors_in_parities:
                errors[key] = set()
                for position in self.get_indices(key, error_index):  #type: ignore
                    errors[key].add(position)
        if len(errors.keys()) < self.n_parities: return set()
        return set.intersection(*errors.values())

    def fix_errors(self):
        if self.errors == set(): return self.word
        matrix = self.matrix.copy()
```

```
        for x, y in self.errors:
            #print(~matrix[x][y])
            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 = 24
    rows, cols = 4, 6
    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, _ = add_errors(word2send, num_errors)
    print("Слово с ошибками:", word2send)

    code2receive = IterativeCodeReceive(length, rows, cols, num_parities, word2send)
    print(code2receive)
```

```
2_2 import numpy as np
from Lab_2_1 import IterativeCode, IterativeCodeReceive, IterativeCodeSend, add_errors

class IterativeCode3D:
    def __init__(self, length, x, y, z, n_parities):
        self.length = length
        self.x, self.y, self.z = x, y, z
        self.n_parities = n_parities
        self.slicis = [
            IterativeCodeSend(
                length // z, x, y, n_parities if n_parities < 5 else n_parities - 1
            ) for _ in range(z)
        ]
        self.word = np.concatenate([slice.word for slice in self.slicis])
        self.matricis = [slice.matrix for slice in self.slicis]
        self.parities = self.combine_parities([slice.parities for slice in self.slicis])
        if n_parities == 5:
            self.parities['z'] = self.calculate_z_parity()  #type: ignore

    def combine_parities(self, parities):
        combined_parities = {key: [] for key in parities[0].keys()}
        for d in parities:
            for key in d:
                combined_parities[key].append(d[key])
        for key in combined_parities:
            combined_parities[key] = np.array(combined_parities[key])  #type: ignore
```

```
        return combined_parities

    def calculate_z_parity(self):
        z_parity = np.zeros((self.x, self.y), dtype=int)
        for matrix in self.matricis:
            z_parity += matrix
        z_parity %= 2
        return z_parity

    def get_indices(self, str, index, z):
        match str:
            case 'row':
                return self.get_row_indices(index, z)
            case 'col':
                return self.get_col_indices(index, z)
            case 'diag_down':
                return self.get_diagonal_indices_down(index, z)
            case 'diag_up':
                return self.get_diagonal_indices_up(index, z)
            case 'z':
                return self.get_z_parity_indices(x=index, y=z)

    def get_row_indices(self, row_index, z):
        return [(row_index, col_idx, z) for col_idx in range(self.matricis[z].shape[1])]

    def get_col_indices(self, col_index, z):
        return [(row_idx, col_index, z) for row_idx in range(self.matricis[z].shape[0])]

    def get_diagonal_indices_up(self, parity_index, z):
        rows, cols = self.matricis[z].shape
        offset = parity_index - (rows - 1)
        diag = np.diagonal(self.matricis[z], offset=offset)
        if offset >= 0:
            return [(i, i + offset, z) for i in range(len(diag))]
        else:
            return [(i - offset, i, z) for i in range(len(diag))]

    def get_diagonal_indices_down(self, parity_index, z):
        indices = self.get_diagonal_indices_up(parity_index, z)
        return [(self.x - 1 - index[0], index[1], z) for index in indices]

    def get_z_parity_indices(self, x, y):
        return [(x, y, z) for z in range(self.z)]

class IterativeCode3DSend(IterativeCode3D):
    def combine_parities_and_word(self):
        array = []
        for slice in self.slicis:
            array.append(slice.combine_parities_and_word())
        if self.n_parities == 5:
            array.append(self.parities['z'].flatten())  #type: ignore
        return np.concatenate(array)

class IterativeCode3DReceive(IterativeCode3D):
    def __init__(self, length, x, y, z, n_parities, word):
```

```
            super().__init__(length, x, y, z, n_parities)
            self.unpack(word)

            self.errors = self.find_errors()
            print(self.errors)
            print("Исправленное слово: ", self.fix_errors())

        def unpack(self, word):
            if self.n_parities == 5:
                z_parities = word[-(self.x * self.y):].reshape(self.x, self.y)
                word = word[:-(self.x * self.y)]
            split_word = np.array_split(word, self.z)
            split_word = [arr.tolist() for arr in split_word]

            slicis = [
                IterativeCodeReceive(
                    self.length // self.z,
                    self.x,
                    self.y,
                    self.n_parities if self.n_parities < 5 else self.n_parities - 1,
                    np.array(split_word[i])
                ) for i in range(self.z)
            ]

            self.word = np.concatenate([slice.word for slice in slicis])
            self.matricis = [slice.matrix for slice in slicis]
               self.current_parities = self.combine_parities([slice.current_parities for slice in
slicis])
            self.parities = self.combine_parities([slice.parities for slice in slicis])
            if self.n_parities == 5:
                self.current_parities['z'] = z_parities
                self.parities['z'] = self.calculate_z_parity()  #type: ignore

        def find_errors(self):
            errors = {}
            for key in list(self.parities.keys()):
                errors[key] = set()
                for i in range(self.z):
                                errors_in_parities = (np.where(self.current_parities[key][i] !=
self.parities[key][i])[0]).tolist()
                    for error_index in errors_in_parities:
                        for position in self.get_indices(key, error_index, i):  #type: ignore
                            errors[key].add(position)
            print(errors)
            if len(errors.keys()) < self.n_parities: return set()
            return set.intersection(*errors.values())

        def fix_errors(self):
            if self.errors == set(): return self.word
            matricis = self.matricis.copy()
            for x, y, z in self.errors:
                matricis[z][x][y] ^= 1
            return np.array(matricis).flatten()


    code2send = IterativeCode3DSend(24, 6, 2, 2, 5)
```

```
word2send = code2send.combine_parities_and_word()
print("Слово с паритетами: ", word2send)
word2send, _ = add_errors(word2send, 5)
print("Слово с паритетами и ошибками: ", word2send)
print()
print("Слово: ", code2send.word)

code2receive = IterativeCode3DReceive(24, 6, 2, 2, 5, word2send)
```

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

Слово с паритетами: [1 1 0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0
 1 0 1 1 0 0 0 1 1 1 0 1 0 1 0]
Слово с ошибками: [1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0
 1 0 1 1 0 0 0 0 1 1 0 1 0 1 0]
Слово: [1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0]
Матрица:
 [[1 1 0 0 0 0]
 [0 1 1 1 1 1]
 [1 1 1 1 0 1]
 [0 0 1 0 1 0]]
Паритеты строк: [0 1 1 0]
Паритеты столбцов: [0 1 1 0 0 0]
Паритеты диагонали (вниз): [1 1 0 0 0 1 1 0 0]
Паритеты диагонали (вверх): [0 1 0 1 0 1 0 1 0]
Найденные ошибки: {(2, 1)}
Исправленное слово: [1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 0]
```

```
Слово с паритетами:  [0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 1 1 0 1 0 1 0 0 0 1 0
 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 0 0 0 1 0
 0 0 1 1 0 0]
Слово с паритетами и ошибками:  [0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 1 0
 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 0 1 0 0 1 1
 0 0 1 1 0 0]

Слово:  [0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 0 0 0 0 1 1]
{'row': {(2, 0, 1), (2, 1, 1)}, 'col': {(1, 0, 1), (4, 0, 1), (5, 0, 1), (2, 0, 1), (3, 0, 1), (0, 0, 1)}, 'diag_down': {(2, 0, 1), (3, 1, 0), (1, 1, 1), (4, 0, 0)},
'diag_up': {(2, 1, 0), (5, 0, 1), (1, 0, 0), (2, 0, 1), (3, 1, 1)}, 'z': set()}
set()
Исправленное слово:  [0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 0 0 0 1 1]
```

Вывод:разработали приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов