

International University of Information Technology

Department of Computer Engineering

**Laborotoy Work №9**

Complete by a student of the group: Ospan Ramazan it2-2310

Checked by senior lector of the Department of Computer Engineering:

Daurenbaeva Nurkamilya

Almaty 2024

Hamming codes are used to detect and correct single-bit errors in transmitted data. This technique adds "parity bits" to information bits, creating a code that allows error localization and correction.

**How It Works**

1. **Structure**:
   1. Data bits are interspersed with parity bits.
   2. Parity bits are placed at positions that are powers of 2 (e.g., 1, 2, 4, 8...).
   3. Parity bits check specific groups of data bits to ensure their sum is even.
2. **Encoding**:
   1. Calculate each parity bit based on the values of the data bits it controls.
   2. Combine parity and data bits into a single sequence.
3. **Decoding**:
   1. Receiver checks all parity bits.
   2. If a parity bit detects an error, it marks the affected positions.
   3. Adding these positions reveals the location of the faulty bit.
   4. The faulty bit is flipped to correct the error.

**Example**

For the data byte 11010010:

* Add 4 parity bits at positions 1, 2, 4, and 8.
* Calculate their values to ensure even sums.
* Resulting code: 011010110010.

If an error occurs, e.g., 011000110010, parity checks identify the error in bit 5. Correcting this restores the original code.

Simple

* Simple error correction for single-bit errors.
* Low redundancy (additional bits are minimal).

**Limitations**

* Cannot detect or correct errors in multiple bits.
* Requires extra memory for parity bits.

Algorithm:

def calculate\_parity\_bits\_length(data\_bits\_length):

# Вычисляем количество контрольных битов

parity\_bits = 0

while (2 \*\* parity\_bits) < (data\_bits\_length + parity\_bits + 1):

parity\_bits += 1

return parity\_bits

def create\_hamming\_code(data\_bits):

# Определяем количество контрольных битов

m = len(data\_bits)

r = calculate\_parity\_bits\_length(m)

# Создаем массив для кодового слова

hamming\_code = ['0'] \* (m + r)

# Заполняем информационные биты, игнорируя позиции контрольных битов

j = 0

for i in range(1, len(hamming\_code) + 1):

if not (i & (i - 1)) == 0: # если i не степень двойки

hamming\_code[i - 1] = data\_bits[j]

j += 1

# Заполняем контрольные биты

for i in range(r):

position = 2 \*\* i

parity = 0

for j in range(1, len(hamming\_code) + 1):

if j & position and j != position:

parity ^= int(hamming\_code[j - 1])

hamming\_code[position - 1] = str(parity)

return ''.join(hamming\_code)

def check\_hamming\_code(hamming\_code):

n = len(hamming\_code)

r = calculate\_parity\_bits\_length(n)

error\_position = 0

for i in range(r):

position = 2 \*\* i

parity = 0

for j in range(1, n + 1):

if j & position:

parity ^= int(hamming\_code[j - 1])

if parity != 0:

error\_position += position

if error\_position == 0:

print("Ошибок нет.")

corrected\_code = hamming\_code

else:

print(f"Ошибка в бите {error\_position}")

# Исправляем ошибку

hamming\_code\_list = list(hamming\_code)

hamming\_code\_list[error\_position - 1] = '1' if hamming\_code\_list[error\_position - 1] == '0' else '0'

corrected\_code = ''.join(hamming\_code\_list)

print("Исправленный код:", corrected\_code)

# Возвращаем код с исправлением ошибки

return corrected\_code

data = "11010010" # Ваши информационные биты

hamming\_code = create\_hamming\_code(data)

print("Код Хемминга:", hamming\_code)

# Проверка кода с возможной ошибкой

received\_code = "11010000" # Пример код с ошибкой

check\_hamming\_code(received\_code)

Work test with one sample mistake:

