| Telekomunikacja - laboratorium | | | | | | Studia zaoczne - inzynierskie | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Nazwa zadania** | | **Kody wykrywające i korygujące błędy transmisji** | | | | | | |
| **Dzień** | **21.04.2022** | | | **Godzina** | **12:00** | | **Rok akademicki** | **21/22** |
| **Imię i Nazwisko** | | | **Bartosz Drągowski** | | | | | |
| Imię i Nazwisko | | | **Szymon Habrych** | | | | | |
| **Opis programu, rozwiązania problemu.** | | | | | | | | |
| Załączony do sprawozdania program posiada TUI dające możliwość wyboru kodowania i dekodowania wiadomości zawartej w pliku.  Kodowanie pliku wejściowego podwaja liczbę bajtów w pliku wyjściowym(każdy bajt oryginalnej wiadomości jest zamieniany na 2 zakodowane bajty). Rezultat operacji kodowania jest umieszczany w wcześniej wskazanym pliku wyjściowym. Plik wejściowy dla operacji dekodowania musi zawierać parzystą liczbę bajtów. Każde wczytane dwa bajty pliku są dekodowane, a potem pojedynczy bajt umieszczany jest w pliku wyjściowym.  Kodowanie:  - Realizowane za pomocą funkcji encodeByte(one\_byte)  - Pojedynczy bajt wiadomości wejściowej zamieniamy na macierz kolumnową i mnożymy ją przez macierz H.  - W każdej parze bajtów, starszy bajt zakodowanej wiadomości jest kopią oryginalnego bajtu wiadomości, natomiast młodszy to rezultat mnożenia tego bajtu przez macierz H (bity parzystości).  Dekodowanie:  - Przeprowadzamy korekcję błędu, jeżeli taki wystąpił.  - W każdej parze bajtów, starszy bajt jest dodawany do zdekodowanej wiadomości.  Sprawdzanie poprawności zakodowanej pary bajtów:  - Realizowane przez correctByte(coded\_byte)  - Jeśli macierz błędu dla wiadomości nie zawiera żadnych jedynek, to znaczy że nie wystąpiło przekłamanie. Macierz błędu jest iloczynem macierzy kolumnowej wiadomości i macierzy H.  Korekcja błędu:  - Realizowane przez metody correctOneBit(coded\_byte, error\_matrix) lub correctTwoBits(coded\_byte, error\_matrix). Program obsługuje przekłamanie, które mogło nastąpić na 1 lub 2 bitach.  - Dla błędu 1 bitowego przekłamany bit odpowiada indeksowi kolumny macierzy H, do której macierz błędu jest identyczna. W tym wypadku zamieniamy wartość bitu w starszym bajcie na przeciwną.  - Dla błędu 2 bitowego sumowane zostają wartości dwóch kolumn macierzy H a następnie porównywane są z wartościami macierzy błędu. Jeżeli taka suma kolumn jest równa macierzy błędu zamieniane zostają bity o indeksach sumowanych kolumn na wartości przeciwne.  - Jeżeli metoda do tej pory nie znalazła błędu, nastąpiło przekłamanie na 3, lub więcej bitach wiadomości i wtedy zostanie rzucony wyjątek i wypisana informacja dla użytkownika o tej sytuacji. | | | | | | | | |
| **Najważniejsze elementy kodu programu z opisem.** | | | | | | | | |
| # wczytuje plik jako bajty i zakoduje # zapisuje wynik do encoded.txt def encode():  input\_file = open("message.txt", 'rb').read()  encoded\_file = c.encodeBytes(bytearray(input\_file))  output\_file = open("encoded.txt", 'wb')  output\_file.write(encoded\_file)   # wczytuje plik jako bajty, poprawia bity i dekoduje # zapisuje wynik jako decoded,txt def decode():  input\_file = open("encoded.txt", 'rb').read()  encoded\_file = c.correctBytes(bytearray(input\_file))  encoded\_file = c.decodeBytes(encoded\_file)  output\_file = open("decoded.txt", 'wb')  output\_file.write(encoded\_file)  # macierz, która spełnia dwa warunki: # 1. nie ma powtarzających się wierszy # 2. żaden wiersz nie jest równy sumie dwóch innych wierszy H\_matrix = np.array((  [1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],  [1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],  [1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],  [1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],  [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],  [0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0],  [1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1] ))   # czyta bajty i szyfruje każdy z nich (encode\_byte) def encodeBytes(bytes):  result = bytearray()  for byte in bytes:  bits = byteToBits(byte)  encoded\_byte = bytearray(encodeByte(bits).tobytes())  result.append(encoded\_byte[0])  result.append(encoded\_byte[1])   return result   # czyta co drugi bajt def decodeBytes(encoded\_bytes):  return encoded\_bytes[::2]   # odczyta dwa bajty - bajt informacji i bajt chroniący # łączy to w bitarray # poprawi uszkodzone bity w bitarray (correct\_byte) # i doda je na koniec wyniku def correctBytes(encoded\_bytes):  result = bytearray()  for i in range(0, len(encoded\_bytes), 2):  byte\_as\_bits = encodedByteToBits(encoded\_bytes[i], encoded\_bytes[i + 1])  corrected\_byte = bytearray(correctByte(byte\_as\_bits).tobytes())   result.append(corrected\_byte[0])  result.append(corrected\_byte[1])   return result   # zmieni bajty na bity def byteToBits(byte):  result = ''  for i in range(8):  result += str(byte % 2)  byte = int(np.floor(byte / 2))  return bitarray(result[::-1])   # zmieni dwa bajty na jeden bitarray def encodedByteToBits(byte\_one, byte\_two):  byte\_as\_bits = bitarray()   bits\_one = byteToBits(byte\_one)  bits\_two = byteToBits(byte\_two)   for j in range(8):  byte\_as\_bits.append(bits\_one[j])   for j in range(8):  byte\_as\_bits.append(bits\_two[j])   return byte\_as\_bits   # zaszyfruje bajty poprzez dodanie bitu parzystości: # weźmie tyle wierszyz H\_matrix ile jest bitów parzystości # i stworzy z nich nową macierz # new\_array jest tablicą bitów(zmienionych z tablicy bitów) # liczy bit parzystości: # (new\_matrix \* T(new\_array)) % 2 # na koniec łączy oryginalny bajt z bitem parzystości def encodeByte(one\_byte):  new\_matrix = H\_matrix[0:, :8]  new\_array = bitarrayToNparray(one\_byte)   parity\_byte = new\_matrix.dot(new\_array.T)  parity\_byte %= 2   encoded\_byte = bitarray()  for i in range(8):  encoded\_byte.append(new\_array[i])   for i in range(8):  encoded\_byte.append(parity\_byte[i])   return encoded\_byte   # zmieni bitarray na np.array który będzie zawierał int zamiast bitów def bitarrayToNparray(bits):  result = []  for i in range(len(bits)):  result.append(int(bits[i]))   return np.array(result)   # zwraca prawdę jeśli error\_matrix jest kolumną zer def checkCodedByte(error\_matrix):  return np.count\_nonzero(error\_matrix) == 0   # liczy: H\_matrix \* np.array(coded\_byte) % 2 # zwróci kolumnę równą jednej z macierzy matrix\_H # jeśli jest jakiś uszkodzony bit # jeśli nie, to zwraca kolumnę zer # potem próbuje poprawić error\_matrix def correctByte(coded\_byte):  error\_array = bitarrayToNparray(coded\_byte)  error\_matrix: np.ndarray = H\_matrix.dot(error\_array) % 2   coded\_byte\_copy = coded\_byte.copy()   if checkCodedByte(error\_matrix):  return coded\_byte  if correctOneBit(coded\_byte\_copy, error\_matrix) is not False:  return correctOneBit(coded\_byte\_copy, error\_matrix)  if correctTwoBits(coded\_byte\_copy, error\_matrix) is not False:  return correctTwoBits(coded\_byte\_copy, error\_matrix)  else:  print("Something went wrong")  raise Exception   # sprawdza czy error\_matrix ma odpowiadającą kolumnę w # transponowanej macierzy H\_matrix # jeśli tak jest, to zmini bit na pozycji równej numerowi kolumniy # jeśli nie, to zwróci fałsz, co będzie oznaczało, że # prawdopodobnie jest więcej uszkodzonych bitów def correctOneBit(coded\_byte, error\_matrix):  i = 0  for column in H\_matrix.T:  # sprawdź czy odpowiadające wartości z error\_matrix i kolumny  # są równe  if np.equal(error\_matrix, column).all():  if coded\_byte[i] == 1:  coded\_byte[i] = 0  else:  coded\_byte[i] = 1  return coded\_byte  i += i   return False   # dodaje dwa wiersze H\_matrix, potem wykonuje sum % 2 # żeby upewnić się, że występują tylko wartości bitowe # jeśli error\_matrix i suma sa takie same, # bity w coded\_byte są poprawione na pozycjach column1 i column2 # inaczej zwróci fałsz (prawdopodobnie jest więcej uszkodzonych bitów) def correctTwoBits(coded\_byte, error\_matrix):  i = 0  j = 0  for column1 in H\_matrix.T:  for column2 in H\_matrix.T:  if i == j:  continue   sum = (column1 + column2) % 2  if np.equal(sum, error\_matrix).all():  if coded\_byte[i] == 1:  coded\_byte[i] = 0  else:  coded\_byte[i] = 1  if coded\_byte[j] == 1:  coded\_byte[j] = 0  else:  coded\_byte[j] = 1  return coded\_byte  j += 1  i += 1  j = 0   return False | | | | | | | | |
| **Podsumowanie wnioski.** | | | | | | | | |
| Program pozwala na poprawne wykrywanie i korekcję błędu jednobitowego i dwubitowego.  Pozwala na kodowanie i dekodowanie plików wraz z korekcją przekłamań. | | | | | | | | |