|  |  |
| --- | --- |
|  | |
| **Semestrální projekt č. 5** | |
|  | |
| David Papaj | |
|  | |
|  |  |
| Semestrální projekt  2021 |  |
|  |  |
|  | |

OBSAH

[zadání 3](#_Toc71302041)

[1 popis programu a jeho funkcí 4](#_Toc71302042)

[1.1 Zakódování znaků na čísla 4](#_Toc71302043)

[1.2 Hammingův kód 5](#_Toc71302044)

[2 Demonstrace konkrétního příkladu 8](#_Toc71302045)

[2.1 Kod 8](#_Toc71302046)

[2.2 Výstup 10](#_Toc71302047)

zadání

Vytvořte program, který bude sloužit k zabezpečení kódových slov, jejich kontrole a případné opravě. K těmto úkonům použijte metodologii Hammingova kódu. Relevantní podklady a zdroje jsou uvedeny níže.

Program bude řešit následující:

**Vyslání vybraného znaku, zakódování, následná kontrola a dekódování.**

Program zakóduje vybraný znak ze znakové sady viz. tabulka níže. Následně kódové slovo zabezpečí pomocí Hammingova kódu K(7,4) použitím generující matice ***G****(použít můžete matice uvedené níže)*. V tomto zabezpečeném kódovém slově vygeneruje zvolený počet chyb (volte 0-3 chyby). Takto upravené (poškozené) kódové slovo program následně zkontroluje dle metodologie Hammingova kódu použitím kontrolní matice ***H****(použít můžete matice uvedené níže)*, provede případnou opravu chyby a dekóduje do výstupního znaku. Vstupním parametrem bude tedy vybraný znak z tabulky a počet generovaných chyb v kódovém slově při přenosu.

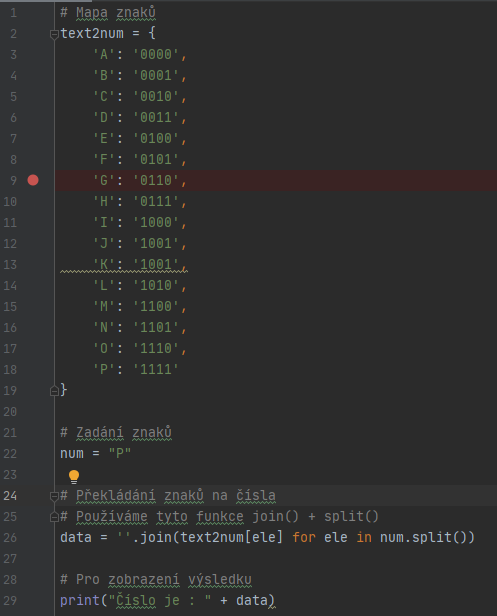
Výstupem (to co se zobrazí po zadání vstupů a odeslání) bude vybraný vstupní znak, jeho odpovídající kódové slovo, použitá generující matice ***G***, zabezpečené kódové slovo vybraného znaku, zvolený počet chyb, přijaté kódové slovo (tvar slova po zanesení chyb), kontrolní matice ***H***, syndrom slova, opravené přijaté kódové slovo, přijatý znak.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Vstupní znak** | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
| **Kódové slovo** | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

popis programu a jeho funkcí

Zakódování znaků na čísla

Pro zakódování znaků využiju Mapa znaků a poté funkci join() a split().



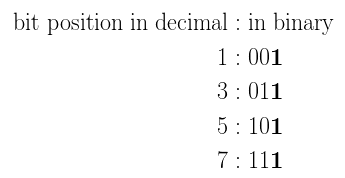


Hammingův kód

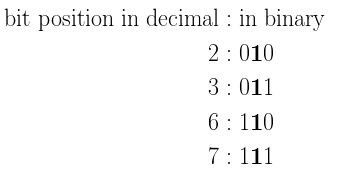
Tedy znak v (7, 4) Hammingovu kódu bude reprezentovat [D 7 , D 6 , D 5 , P 4 , D 3 , P 2 , P 1 ] , kde D představuje informační bity a P představuje paritní bity v příslušných bitové pozice. Dolní indexy označují pozici zleva doprava data a paritní bity. Poznamenáváme, že paritní bity jsou umístěny na pozici, která je mocninou dvou (bitové pozice 1,2,4).

Nyní reprezentujte bitové pozice v binárním formátu.

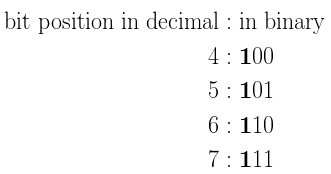
Při pohledu zleva doprava první paritní bit ( P 1 ) pokrývá bity na pozicích, jejichž binární reprezentace má 1 na nejméně významném bitu. Zjistili jsme, že P 1 pokrývá následující bitové pozice



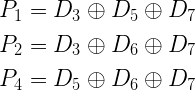
Podobně, druhý bit parity ( P 2 ), se vztahuje na bity v polohách, jejichž binární reprezentace má 1 na druhém nejméně významný bit. Proto P 2 pokrývá následující bitové pozice



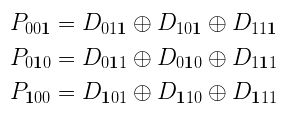
A konečně, je bit třetí parity ( P 4 ) pokrývá bity v polohách, jejichž binární reprezentace má 1 na nejvýznamnějšího bitu. Proto, P 4 zahrnuje následující bitových pozic

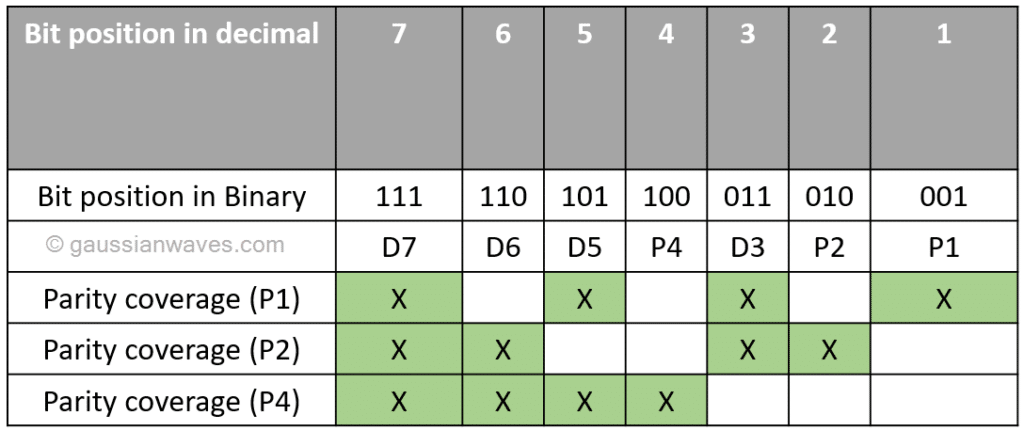


Pokud se u paritních bitů řídíme paritním schématem, musí se počet 1 pokrytých paritními bity přidat k sudému číslu. Což znamená, že XOR bitů krytých paritou (včetně paritních bitů) musí mít za následek 0. Proto platí následující rovnice

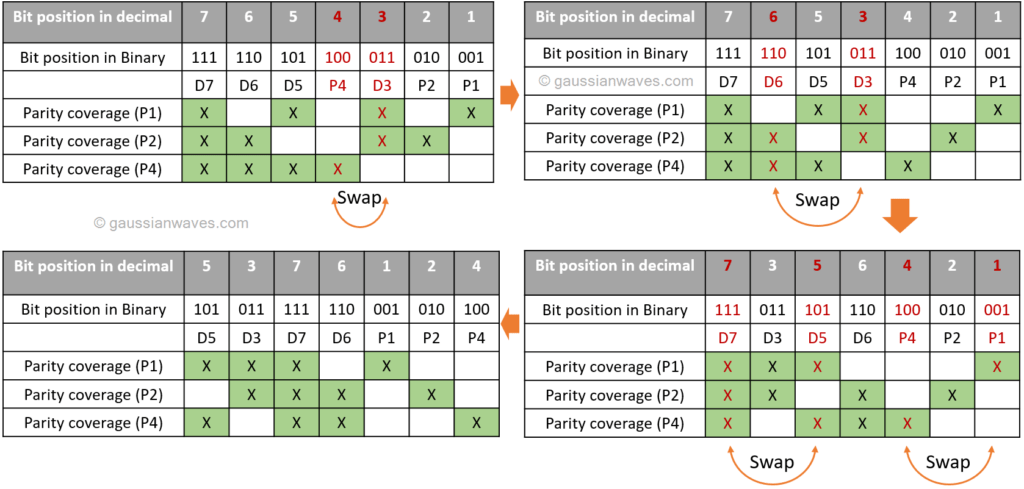
.

Pro přehlednost si představme dolní indexy v binárním formátu.

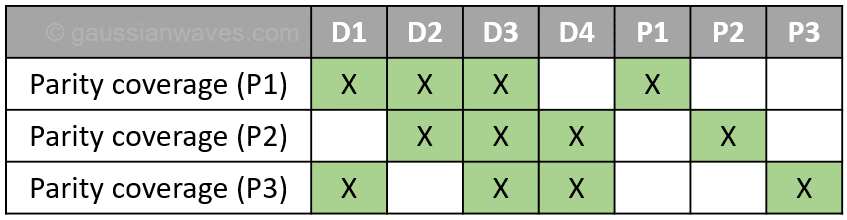


[](https://www.gaussianwaves.com/buy-books/)

Poznamenáváme, že paritní bity a datové sloupce jsou rozptýleny. Toto je příklad nesystémové struktury Hammingova kódu. Můžeme pokračovat v práci na výše uvedeném stole. Nebo můžeme také znovu uspořádat položky této tabulky pomocí elementárních transformací, takže se vykreslí systematický Hammingův kód.

[](https://www.gaussianwaves.com/buy-books/)

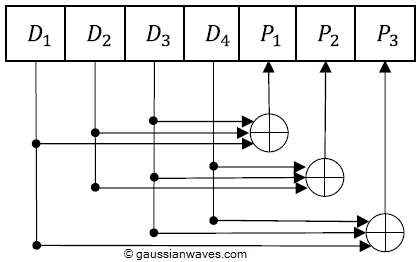
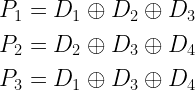
Po opětovném uspořádání sloupců vidíme, že paritní sloupce jsou na konci pěkně spojeny dohromady. Můžeme také zrušit dolní indexy dané paritním / datovým umístěním a znovu je indexovat podle našeho pohodlí. To dává následující strukturu **(7,4)** Hammingovu kódu.

x[](https://www.gaussianwaves.com/buy-books/)

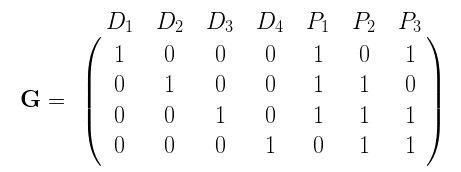
Výše uvedenou systematickou strukturu využiju poté pro procesování kódu.

**Proces kódování**

Vzhledem ke struktuře u obrázku x se paritní bity počítají z následujících lineárně nezávislých rovnic pomocí přídavků modulo-2.

[](https://www.gaussianwaves.com/buy-books/)

Na straně vysílače implementuje kodér Hamming generátorovou matici -G. Je jednodušší sestrojit matici generátoru z lineárních rovnic uvedených ve výše uvedené rovnici. Lineární rovnice ukazují, že informační bit D 1 ovlivňuje výpočet parit v P 1 a P 3 . Podobně informační bit D 2 ovlivňuje P 1 a P 2 , D 3 ovlivňuje P 1 , P 2 a P 3 a D4 ovlivňuje P2 a P3



Maticové operace, kodér přijímá 4 bitovou zprávu blok , vynásobí to maticí generátoru a generuje 7 bitová kódová slova .

Demonstrace konkrétního příkladu

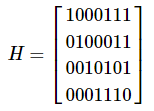
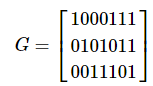
Kod

je schéma dopředné korekce chyb (FEC), které lze použít k detekci a opravě bitových chyb. Bity pro opravu chyb jsou známé jako Hammingovy bity a počet, který je třeba přidat k datovému symbolu, je určen výrazem: 2^n > = ( m + n + 1 )

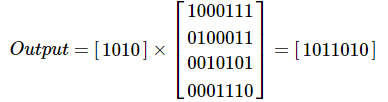
, kde m je počet bitů v datovém symbolu a n je počet Hammingových bitů.

Hammingovy bity se podle potřeby vkládají do znaku zprávy. Obvykle se přidávají na pozice, které jsou mocninami 2, tj. 1., 2., 4., 8., 16. bitová pozice atd. Například pro kódování znaku 011001 pak, počínaje z pravé strany, budou Hammingovy bity vloženy do 1., 2., 4. a 8. bitové pozice. Tato stránka simuluje jednobitovou chybu.

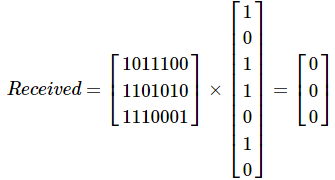
S (7,4) Hammingovým kódem vezmeme 4 bity dat a přidáme 3 Hammingovy bity, abychom dali 7 bitů pro každou 4bitovou hodnotu. Vytvoříme matici generátoru kódu G a matici kontroly parity H. Vstupní data se vynásobí G a pro kontrolu výsledku se vynásobí H:

Pokud máme datový vstup [1 0 1 0], pak k vytvoření zprávy k odeslání dostaneme:



Přenesené bity tedy budou „1 0 1 1 0 1 0“.



který identifikuje, že nejsou žádné chyby.

Využil jsem modulu NumPy.

import numpy  
import sys  
  
# Mapa znaků  
text2num = {  
 'A': '0000',  
 'B': '0001',  
 'C': '0010',  
 'D': '0011',  
 'E': '0100',  
 'F': '0101',  
 'G': '0110',  
 'H': '0111',  
 'I': '1000',  
 'J': '1001',  
 'K': '1001',  
 'L': '1010',  
 'M': '1100',  
 'N': '1101',  
 'O': '1110',  
 'P': '1111'  
}  
  
# Zadání znaků  
num = "P"  
  
# Překládání znaků na čísla  
# Používáme tyto funkce join() + split()  
data = ''.join(text2num[ele] for ele in num.split())  
  
  
if (len (sys.argv) > 1):  
 data = sys.argv[1]  
  
  
def encode(m,g):  
 x = numpy.dot (m,g) % 2  
 return x  
  
  
def decode(m,h):  
 dec = numpy.dot (h,m) % 2  
 return dec  
  
  
def Flip\_Bit(enc,bitpos):  
 if (enc[bitpos] == 1):  
 enc[bitpos] = 0  
 else:  
 enc[bitpos] = 1  
 return enc  
  
  
def hamming(a):  
 g = numpy.array ([[1,0,0,0,1,1,1],[0,1,0,0,0,1,1],[0,0,1,0,1,0,1],[0,0,0,1,1,1,0]])  
 h = numpy.array ([[1,0,1,1,1,0,0],[1,1,0,1,0,1,0],[1,1,1,0,0,0,1],])  
  
 a = a[:4]  
  
 enc = encode (a,g)  
  
 dec = decode (enc,h)  
 print ("Vstup \t\tKod (1 error) \tError pozice")  
 print (a,"\t",enc,"\t",dec)  
  
 print ("----------------------")  
 print ("1 error:")  
 print ("Vstup \t\t Kod (1 error) \t Error pozice")  
 for i in range (0,7):  
 enc = encode (a,g)  
 enc1 = Flip\_Bit (enc,i)  
 dec = decode (enc1,h)  
 print (a,"\t",enc1,"\t",dec)  
  
  
a = numpy.frombuffer (data.encode (),dtype=numpy.uint8) - ord ('0')  
  
hamming (a)

Výstup

