Projektni Zadatak 2

Kompresija i zastita podataka

Projektni zadatak 2

Emilija Ristic 114/2018

Za pokretanje samog programa potrebno pokrenuti komandu u terminalu:
python -X utf8
projektni_zadatak2.py

LDPC

Low-Density Parity-Check kodovi za proveru pariteta niske gustine - linearni kod za ispravljanje grešaka, metodom prenosa poruke preko šumnog kanala za prenos.

Konstrušu se sa retkom (sparse) kontrolnom matricom H.

Regularni LDPC kodovi zadovoljavaju svojstvo da svaki red odnosno kolona matrice H sadrži `w_r`, odnosno `w_c` jedinica.

Matrica H je retka ako važi da je `w_r << m` i `w_c << n`.

Za konstrukciju LDPC koda važe sledeća pravila:

1. Prvih `(n-k)/w_c = w_r` redova matrice H dobijaju se tako što se u svaki red upisuje `w_r` jedinica.
2. I redu jedinice se postavljaju u kolone sa indeksima od 1 do `w_r`
3. II redu jedinice se upisuju u kolone sa indeksima od `w_r +1` do `2w_r`
4. Presotalih `w_c-1` grupa redova dobijaju se permutacijom `w_r+1` redova

Postavka:

$$n = 15, n - k = 9, w_r = 5, w_c = 3$$

```
import random

n = 12
k = 3
wr = 4
wc = 3

# generisanje 2. i 3. grupe redova koristimo standardni generatom sa fiksiranim seed-om (broj indeksa)
random.seed(114,2018)

# upisujemo nule
H = [[0 for _ in range(n)] for _ in range(n - k)]

# popunjavamo 1. grupu
for i in range((n - k) // wc):  # Prvih (n - k) / wc = 3 redova
    start_col = i * wr
    for j in range(wr):
        H[i][start_col + j] = 1

# w_r + 1 do 2w_r
# popunjavanje 3. i 4. grupe
for g in range(1, wc):
    perm = list(range(n))
    random.shuffle(perm)  # permutacija kolona
    for i in range((n - k) // wc):
```

```
        for j in range(n):
            H[g * ((n - k) // wc) + i][j] = H[i][perm[j]]


print("LDPC matrica H:")
for row in H:
    print(' '.join(map(str, row)))
```

Tabela sindroma i korektora i odredjivanje kodnog rastojanja

Tabela sindroma i korektora

```
# funkcija za izračunavanje sindroma
def izracunavanjeSindroma(H, e):
    m = len(H)
    n = len(H[0])
    H_transponovano = [[H[j][i] for j in range(m)] for i in range(n)]
    sindrom = [sum(e[i] * H_transponovano[j][i] for i in range(m)) % 2 for j in range(n)]
    return tuple(sindrom)

# funkcija za generisanje binarnih vektora
def binarniVektori(n):
    interval_vektora = 2 ** n
    vektori = []
    # generisanje vektora greške (korektora)
    for i in range(interval_vektora):
        # pretvaramo i u binarni niz sa n bitova
        vektor = [int(x) for x in format(i, f'0{n}b')]
        vektori.append(vektor)
    print(vektori)
    return vektori

# funkcija za sortiranje vektora prema težini
def sortirajPremaTezini(vektori):
    return sorted(vektori, key=lambda x: sum(x))

# funkcija za generisanje tabele sindroma i korektora
def ispisiSindromTabelu(H):
    binarni_v = binarniVektori(len(H[0]))
    # print("Binarni vektori",binarni_v)
    sortirani_v = sortirajPremaTezini(binarni_v)
    # print("sortirani_v",sortirani_v)
    sindrom_tabela = {}

    for e in sortirani_v:
        sindrom = izracunavanjeSindroma(H, e)
        if sindrom not in sindrom_tabela:
            sindrom_tabela[sindrom] = e

    return sindrom_tabela

sindrom_tabela = ispisiSindromTabelu(H)

print("Sindrom => korektor")
for sindrom, korektor in sindrom_tabela.items():
    print(f"{sindrom} => {korektor}")
```

Kodno rastojanje

$$d(C) = min(dH(x(i), x(j)))$$

Kodno rastojanje predstavlja minimalno Hamingovo rastojanje izmedju bilo koje dve razlicite kodne reci u kodu C.

Hamingovo rastojanje:

$$d_h(x, y)$$

izmedju dve binarne reci `x` i `y` je broj pozicija na kojima se razlikuju.

```
def hamingovoRastojanje(x,y):
    return sum(1 for x_i, y_i in zip(x,y) if(x_i != y_i))

def generisiKodneReci(H):
    n = len(H[0])
    kodne_reci = []

    for i in range(2 ** n):
        bin_rec = [int(b) for b in format(i, f'0{n}b')]
        sindrom =  [sum(bin_rec[j] * H[k][j] for j in range(n)) % 2 for k in range(len(H))]
        if all(s == 0 for s in sindrom):
            kodne_reci.append(bin_rec)
    #print("kodne_reci",kodne_reci)
    return kodne_reci

def kodnoRastojanje(H):
    kodne_reci = generisiKodneReci(H)
    min_rastojanje = float('inf')

    for i in range(len(kodne_reci)):
        for j in range(i + 1, len(kodne_reci)):
            rastojanje = hamingovoRastojanje(kodne_reci[i], kodne_reci[j])
            if rastojanje < min_rastojanje:
                min_rastojanje = rastojanje
    return min_rastojanje




kodno_rastojanje = kodnoRastojanje(H)
print(f"Rastojanje koda: {kodno_rastojanje}")
```

Gallager B dekodiranje

Svrha ovog algoritma je da dekodira LDPC kod. Cilj je da se dekodira primljeni vektor uzimajući u obzir informaciju o sindromu i vrednosti pragova.

```
def gallagerDekodiranje(H,vektor, max_iter=50, th0=0.5, th1=0.5):
    m = len(H)
    n = len(H[0])
    x = vektor[:] # postavljanje dekodiranog vektora

    for iteration in range(max_iter):
        sindrom = izracunavanjeSindroma(H, x)
        if all(s == 0 for s in sindrom):  # ako je sindrom nula, dekodiranje je uspešno
            # print(f"Dekodiranje uspešno nakon {iteration} iteracija.")
            return x
        # na osnovu praga potrebno je da ažuriramo vrednosti
        for i in range(n):
            suma_proveri = sum(H[j][i] * sindrom[j] for j in range(m))
            if suma_proveri < th0:
                x[i] = 0
            elif suma_proveri > th1:
                x[i] = 1
```

```
    print("Dekodiranje nije uspešno nakon maksimalnog broja iteracija.")
    return x

# Funkcija za generisanje greške koja se ne može ispraviti
def generisiNeispravljivuGresku(H, th0=0.5, th1=0.5):
    n = len(H[0])
    greske = sortirajPremaTezini(binarniVektori(n))

    for e in greske:
        r = [e_i for e_i in e]  # vektori greske
        dekodirani_v = gallagerDekodiranje(H, r, max_iter=50, th0=th0, th1=th1)
        sindrom = izracunavanjeSindroma(H, dekodirani_v)

        if any(s != 0 for s in sindrom):
            print(f"Nepopravljiva greška: {e}")
            return e

    return None


neispravljiva_greska = generisiNeispravljivuGresku(H, th0=0.5, th1=0.5)
if neispravljiva_greska:
    print(f"Greška sa najmanje jedinica koju Gallager B algoritam nije uspeo da ispravi: {neispravljiva_greska}")
else:
    print("Sve generisane greške su ispravljive.")
```

```
output:
LDPC matrica H:
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 1 0 1
0 1 0 1 0 1 0 0 0 0 0 1 0 1 0
0 0 0 1 1 1 1 0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 1 1 0 0 1 0 0 1
Sindrom => korektor
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) => [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1) => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0) => [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0) => [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1) => [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0) => [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) => [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1) => [0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1) => [0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0) => [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0) => [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1) => [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1) => [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1) => [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0) => [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0) => [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1) => [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1) => [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0) => [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0) => [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1) => [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0) => [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1) => [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0) => [0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0) => [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1) => [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1) => [0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1) => [0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
(0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0) => [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0) => [0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1) => [0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1) => [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1) => [1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1) => [1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0) => [1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0) => [1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1) => [1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1) => [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1) => [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0) => [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0) => [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1) => [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0) => [0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1) => [0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0) => [0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0) => [0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1) => [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1) => [1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0) => [1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0) => [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1) => [1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1) => [1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1) => [1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0) => [1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0) => [1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1) => [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1) => [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0) => [0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0) => [1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1) => [1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0) => [1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0) => [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1) => [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0) => [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]

Rastojanje koda: 1

Dekodiranje nije uspesno nakon maksimalnog broja iteracija.
Nepopravljiva greska: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Gre□ka sa najmanje jedinica koju Gallager B algoritam nije uspeo da ispravi: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0
, 0, 0]
```