SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Predmet: Teorija informacije
Ak. godina: 2010./2011.

Laboratorijske vježbe:
I Z V J E Š T A J

Grupa {AB31}:
1. {hrckov, 0036xxxxxx}

Zadatak

Zadan je komunikacijski sustav koji se sastoji od izvora informacije, kodera informacije, kodera kanala, kanala na koji utječu smetnje, dekodera kanala, dekodera informacije i odredišta. Izvorište generira slučajni slijed od 10000 simbola u kojem će se simboli pojavljivati s zadanim vjerojatnostima.

p(a) = 0.4, p(b) = 0.1, p(c) = 0.1, p(d) = 0.2, p(e) = 0.2

Te simbole potrebno je kodirati Huffmanovim kodom. Nakon toga, novonastale slijedove simbola (tzv. kodirana poruka) potrebno je kodirati Hammingovim kodom [15,11,3] (dobivamo zaštitno kodiranu poruku). Zaštitno kodirana poruka prenosi se komunikacijskim kanalom u kojem:
i) u jednom slučaju nema smetnji, dok
ii) u drugom slučaju smetnje djeluju na poslane podatke.

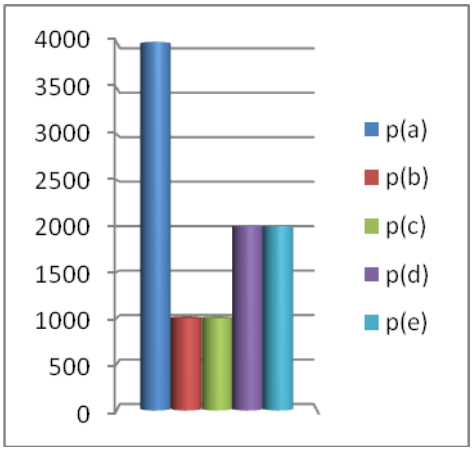
Vjerojatnost pogreške bita u kanalu je 1/20

Podatke koji su prošli kroz komunikacijski kanal potrebno je dekodirati na odredištu.

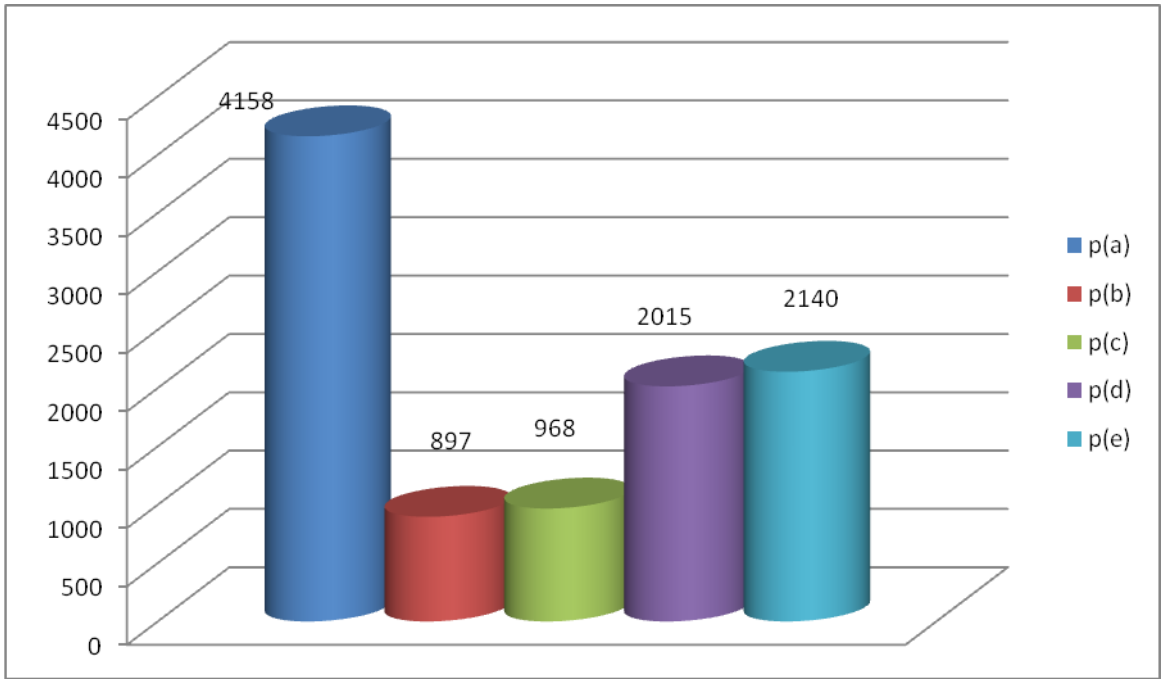
Rješenje

Informacijska svojstva kanala

1. Nacrtajte histogram generiranog slijeda.



Slika 1: Idealno izvorište



Odredite vjerojatnosti pojavljivanja simbola u generiranom slijedu:

$p(a) = 0{,}4158$; $p(b) = 0{,}897$; $p(c) = 0{,}0968$; $p(d) = 0{,}2015$; $p(e) = 0{,}2140$

2. Izračunajte entropiju izvorišnog skupa simbola.

H(izv.)=_2,117721_ [bita/simbolu]

3. Izračunajte entropiju odredišnog skupa simbola u slučaju da se koristi zaštitno kodiranje.

H(odr.)=_2,106267_ [bita/simbolu]

4. Izračunajte transinformaciju u slučaju da se koristi zaštitno kodiranje:

I(*X*;*Y*) =___/___ [bita/simbolu]

Dolazi do raspada kanala u 99% slučajeva ako koristimo kanal sa vjerojatnosti pogreške 1/20. Inače, ako je kanal sa vjerojatnosti pogreške p=0: I(X;Y)=H(izv.).

5. Izračunajte entropiju odredišnog skupa simbola u slučaju da se NE koristi zaštitno kodiranje.

H(odr.)=_2,099766_ [bita/simbolu]

6. Izračunajte transinformaciju u slučaju da se NE koristi zaštitno kodiranje:

I(*X*;*Y*) =___/___ [bita/simbolu]

Dolazi do raspada kanala u 99% slučajeva ako koristimo kanal sa vjerojatnosti pogreške 1/20. Inače, ako je kanal sa vjerojatnosti pogreške p=0: I(X;Y)=H(izv.).

Zaštitno kodiranje

Napomena: Sve niže definirane vrijednosti je generiranjem velikog broja ulaznih simbola potrebno odrediti statistički. Ako možete odrediti tu vjerojatnost računski, tada provjerite da li rezultat računa odgovara rezultatu dobivenom statističkom analizom.

1. Statistički dobivena udaljenost za zadani zaštitni kôd iznosi: $d(K) = 3$

2. Računski dobivena udaljenost za zadani zaštitni kôd iznosi $d(K) = 3$

3. Kôd može ispraviti __1__ pogrešaka.

4. Kôd može detektirati __2__ pogrešaka*.
   *kod ne razlikuje da li je došlo do jedne ili dvije pogreške, te pokušava ispraviti dvostruku pogrešku što rezultira krivim dekodiranjem.

5. Navedite primjer u kojem zaštitni kôd ne bi mogao ispraviti niti detektirati pogrešku:

   Ako bi se svi bitovi u kanalu invertirali, dekoder ne bi mogao ispraviti niti detektirati pogrešku.

6. Vjerojatnost ispravnog dekodiranja poruke iznosi (statistički dobiven podatak):
   P(K)=0,82905.

7. Kodna brzina zaštitnog kôda iznosi: $R = \frac{k}{n} = 0{,}733$

8. Navedite generirajuću matricu **G**:

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

9. Navedite matricu provjere pariteta **H** za zadani kôd:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Entropijsko kodiranje

1. Prosječna duljina kodne riječi: $L = 2.2$

2. Za zadani kôd navesti postupak kodiranja zadanog izvorišta. Ako se kodiranje temelji na rječniku, tada je potrebno kodirati proizvoljni niz simbola. U oba slučaja potrebno je postupak potkrijepiti detaljnim komentarima.

Zadani kod je Huffmanov kod. U teoriji on se generira stvaranjem binarnog stabla, pa smo ga tako i programski implementirali. Samo kodiranje se temelji na promatranju ulaznog niza znak po znak i zamjenjivanju znaka sa generiranim binarnim kodom. Samo kodiranje nije se pokazalo kao problem pa ga nećemo detaljno objašnjavati, nego ćemo detaljnije objasniti ostvarenje generatora koda.

Generator ima već postavljene vjerojatnosti pojavljivanja simbola kako bi bio jednoznačno generiran kod i u koderu i u dekoderu (koristimo varijablu ulaz_vjero). Idući korak u generiranju koda je stvaranje binarnog stabla. Svakom čvoru u stablu dajemo određene vrijednosti: vrijednost (element), vjerojatnost, cvor1, cvor2 (nizi cvorovi), kod (dodjeljeni kod cvoru – 0 ili 1). Stvaranje binarnog stabla započinje od listova prema korjenu. Za svaki zadani element stvaramo čvor i dodajemo mu poznate vrijednosti: vjerojatnost i naziv (vrijednost). Sve čvorove stavljamo u listu čvorova pomoću koje im pristupamo.

Generacija Huffmanovog koda se temelji na povezivanju čvorova sa najmanjom vjerojatnosti u nadčvor i pritom određujemo "kod" tih čvorova. To se ponavlja dok god ne postoji samo jedan slobodni čvor – korjen stabla, sa vjerojatnosti 1. To radi i naš generator.

Ponavlja sljedeći postupak:

1. Pronalazi dva čvora sa najmanjim vjerojatnostima (koristeći funkciju minimumi_rjecnika (rjecnik)

2. Stvara novi čvor i dodaje ga u listu

3. Novom čvoru za vjerojatnost postavi zbrojene vjerojatnosti minimalnih čvorova, vrijednost spojene vrijednosti čvorova, cvor1 i cvor2 njihove vrijednosti

4. Minimalnim čvorovima postavi vrijednosti kod na 0 (čvor sa manjom vjerojatnosti), odnosno na 1 (čvor sa većom vjerojatnosti).
5. Izbriše minimalne čvorove iz liste

Vrijedi napomenuti da koristimo dodatnu listu sa spremanje svih čvorova kako nebi izgubili čvorove pri izgradnji stabla.

Postupak se ponavlja dok god vjerojatnost novog čvora ne postane 1.

Sada kada je izrađeno binarno stablo generator započinje sa prolazom kroz stablo od korjena prema listovima pritom "skupljajući" kod po čvorovima (ostvareno rekurzivnom funkcijom kod_trazi(stablo,trenutni_cvor,trenutni_kod)). Započevši od korjena, generator ispituje čvor da li ima podčvor, ako ima sprema kod tog čvora i ispituje podčvorove sve dok ne dođe do listova nakon čega za određeni list (element) ima sakupljeni kod (ostvareno rekurzivnim spustom kroz stablo – za svaki čvor osim lista ponovo se poziva funkcija kojoj se prosljeđuje primljeni kod i kod trenutnog čvora, list samo dodaje svoj kod i sprema u listu kodova svoju vrijednost i svoj kod).

Po završetku spušta se kroz stablo i imamo listu elemenata i njihovih kodova. Koder učitava ulazni niz i za svaki element u nizu zapisuje kod tog elementa.

Kod za vjerojatnosti i znakove zadane u zadatku je: a=0, b=1100, c=1101, d=111, e=10

Dodatak

Uvodna napomena:
Labos je pisan i testiran koristeći Python 2.6.5 na sustavu Ubuntu 10.04, te Python 2.7 na sustavu Windows 7. Na nekim drugim verzijama program nije radio kako treba, tako da smo uz izvorni kod dodali i verziju kompajliranu u bytecode na računalu na kojem je radilo za slučaj da bude problema pri pokretanju programa.

Zadaci za vježbu su bili poprilično korisni i zanimljivi, iako na dijelovima kompliciranu za izvesti. Najveći problem pri izradi labosa se pokazao hammingov (de)koder, iz razloga sto riječ koju kodiramo mora imati točno 11 znakova, no najčešće zadnja riječ ima manje, pa kako ju ne bi odbacivali implementirali smo način produživanja rijeci nulama i označavanja zadnjeg bitnog znaka kako bi dekoder mogao ispravno dekodirati.
Cijeli kod smatramo dobro napisanim i komentiranim, no ipak ćemo neke dijelove koda ovdje spomenuti.
Rad huffmanovog kodera smo već objasnili u za to predviđenom dijelu, tako da ćemo ovdje objasniti rad zaštitnog kodera – hammingovog kodera.
Prije nego sto započne sa radom koder mora izraditi generirajuću matricu G. Matrica G se stvara koristeći paritetnu matricu H tako da rad programa započinje izgradnjom nje.

```
for i in range(0,k):
    s=0
    g=0
    for j in range(0,n-1):
        if((j+1==2**i) and (s==0)):
            l.append(1)
            s=1
            g=1
        elif(s==0):
            l.append(0)

        if((s==1) and (g<2**i)):
            l.append(1)
            g+=1
        elif((s==1) and (g<2*(2**i))):
            l.append(0)
            g+=1
        elif(s==1):
            l.append(1)
            g=1
    h.append([])
    h[i].extend(l)
    l=[]
```

Sto se u stvari događa? Program stvara matricu čiju veličinu određuje tip hammingovog koda koji zadamo, u tu matricu dodaje red po red 0 i 1, pritom svaki red ima 2^red ponavljanja 0 odnosno 1.
Nakon toga započinje uređivanje matrice H dok ne dobijemo G.
Prvo pobrišemo sve stupce na lokacijama potencije broja 2.

```
g=[]
x=len(h[0])
for i in range (0,len(h)):
    s=0
    for j in range(0,len(h)):
```

```
        del h[i][(2**j)-s-1]
        s+=1
```

Nakon čega preostale redove postavimo na lokacije potencija broja 2, a ostatak matrice popunimo jediničnom matricom.

```
l=[]
for i in range (0,x):
    l.append([])
for i in range (0,len(h)):
    l[2**i-1].extend(h[i])
s=0
for i in range (0,len(l)):
    if(l[i]==[]):
        for j in range (0,len(l[0])):
            l[i].append(0)
        l[i][s]=1
        s+=1
```

Tim postupkom smo stvorili transponiranu matricu G^t, pa je samo preostalo transponiranje dobivene matrice.

```
for i in range (0,len(l[0])):
    g.append([])
    for j in range (0,len(l)):
        g[i].append(l[j][i])
```

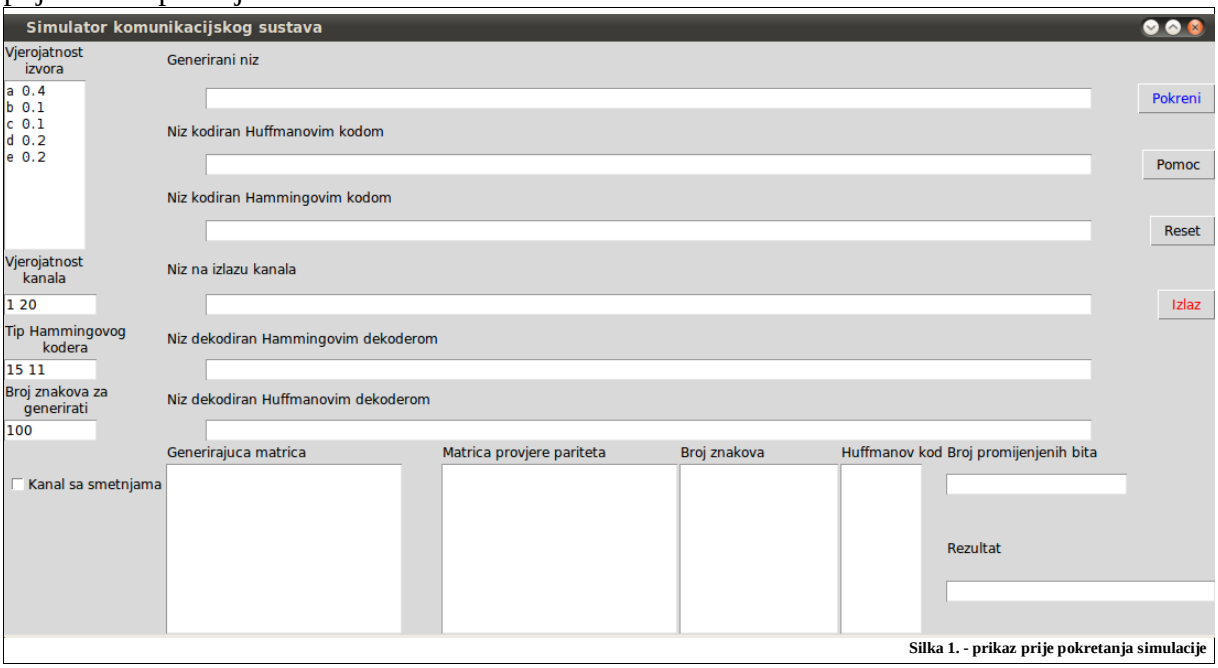
Time završava postupak generiranja matrice G i koder može započeti sa radom.
Koder čita ulazni niz znak po znak i sprema ih u varijablu y, sve dok varijabla y nije potrebne duljine (definirana tipom hammingovog kodera), nakon čega poziva funkciju mnozenje_matrica() koja množi varijablu y sa matricom G cime dobivamo kodirani niz. U slučaju da y bude kraci od potrebne veličine, koder pamti lokaciju zadnjeg pročitanog znaka, ostatak varijable y puni sa 0 i kodira, nakon čega dodaje znak '&' na mjesto nakon zadnjeg pročitanog znaka kako bi dekoder znao koje bitove da odbaci nakon dekodiranja.

```
for z in x:
#kod kraci nego potrebno
    if (z=='#'):
        s=len(y)
        y+='0'*(l-len(y))
        mnozenje_matrica()
        m=0
        n=0
        for k in range(0,len(kod)):
            if (k==(2**m-1)):
                izlaz.write(kod[k])
                m+=1
            elif (n==s):
                izlaz.write("&"+kod[k])
                n+=1
            else:
                izlaz.write(kod[k])
                n+=1
        izlaz.write('#')
        break
#kod nije kraci
    if s!=l:
        y+=z
        s+=1
    else:
```

```
        mnozenje_matrica()
        for k in kod:
            izlaz.write(k)
        y=z
        s=1
        del kod
        kod=[]
```

Dodatak*

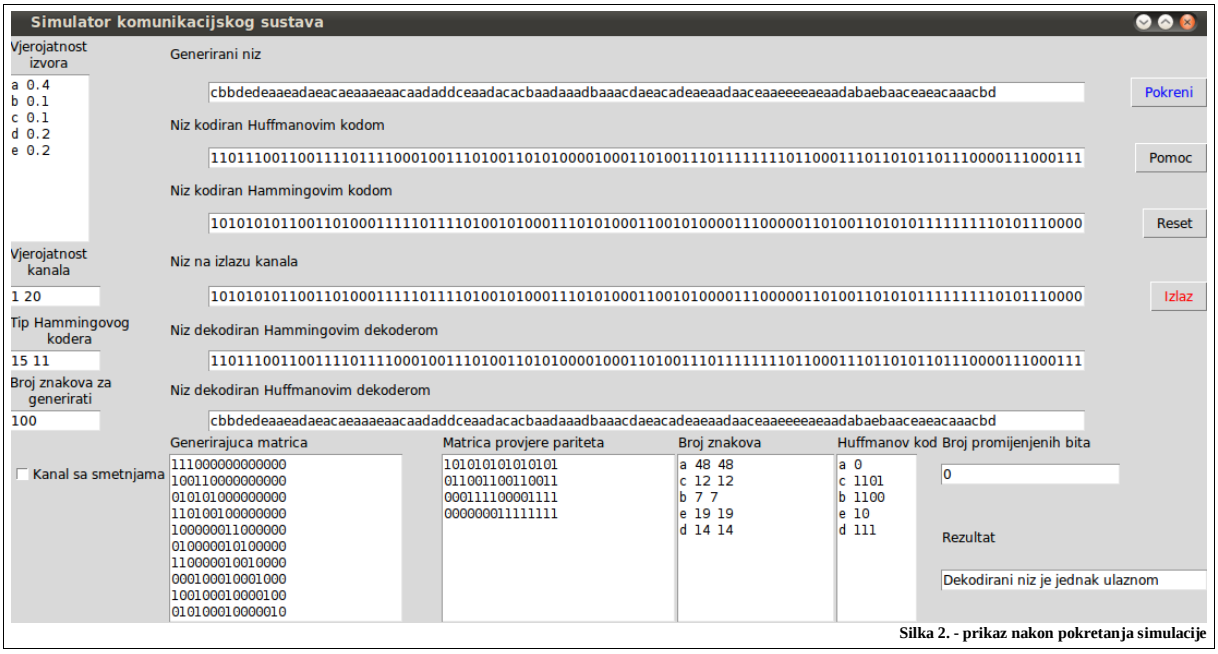
Dodatno kao dio labosa odlučili smo izraditi verziju programa koja se lakše koristi, kod koje se vrijednosti svih elemenata lakše mijenjaju itd, te smo izradili grafičko sučelje za nas program, također, radi kvalitetnijeg rada smo odbacili komunikaciju datotekama, i usvojili objektno orijentirani pristup rada, pri čemu je svaki element sustava objekt i komunicira sa drugim varijablama. Isto tako, ne računa vektor pogreške nego nasumično odabire da li ce pojedini bit promijeniti ili ne itd.



Silka 1. - prikaz prije pokretanja simulacije

Sučelje je napravljeno da ga se lako koristi i da sadrži sto vise informacija, nažalost, nije sve 100% gotovo jer nije bilo vremena dodati sve stvari poput entropije i transinformacije u ovu verziju programa, ali nadamo se to u budućnosti napraviti, kao i unošenje vlastitih vrijednosti u pojedine dijelove programa i njihovo zasebno pokretanje.

Program se pali sa već definiranim vrijednostima za nas zadatak, no te vrijednosti su promjenjive. Pritiskom na gumb Pokreni, započinje simulacija definiranog sustava. Ponovnim pritiskom na Pokreni započinje nova simulacija.



Silka 2. - prikaz nakon pokretanja simulacije

Zaključak

Iako nam se na prvi pogled zadatak činio kompliciranim i teškim, nakon početka rada i detaljnijeg proučavanja teorije sve je sjelo na svoje mjesto i dobili smo smisleni i kvalitetni program koji funkcionira za razne slučajeve, ne samo za onaj zadan na početku. Najveći problem je predstavljalo "fino ugađanje" hammingovog kodera i dekodera, no sve je uspješno napravljeno.