**Poročilo za laboratorijsko vajo 3 pri predmetu Informacija in kodi**

Jure Špeh, 64170334

Sodelavec: Matic Sedej

Kazalo:

## 1. Uvod

Pri tej vaji smo se spoznali z gospodarnim kodiranjem, ki pomaga pri bolj učinkovitem zapisu sporočil. Datoteko zapišemo z manj bajti, ohranimo pa informacijsko vsebino. Postopek imenujemo tudi kompresija datotek.

Spoznali smo postopek LZW, pri katerem sproti gradimo slovar podnizov sporočila.

## 2. Naloga 1

V nalogi 1 smo preučili delovanje algoritma za kodiranje. Pri tem smo si pomagali s predstavitvijo vaje in funkcijo *kompresiraj* sproti komentirali.

Izdelati je bilo potrebno tudi funkcijo za dekodiranje *dekompresiraj* s katero nato dekodiramo prej zakodirano datoteko. Funkcija sprejme kompresijsko kodo KT in sestavi osnovni inverzni slovar znakov PT z 256 vrednostmi, ki ga nato sproti dopolnjujemo. Nato začnemo z branjem kodov iz vhodne tabele KT. Prvi kod poiščemo v slovarju PT in vrednost zapišemo v tabelo ST.

Nato v for zanki nadaljujemo z dekodiranjem. V primeru, da se kod že nahaja v slovarju PT, vrednost zapišemo v ST in sestavimo nov niz, ki ga dodamo na naslednje prosto mesto v slovarju PT. Nov niz je sestavljen iz prejšnjega niza in vrednosti trenutnega niza *s*.

```
if c in PT:
    # Poiscemo vrednost za kljuc c
    s = PT[c]
    # Dodamo vrednost v tabelo ST
    ST.append(s)
    # Sestavimo novo vrednost iz prejšnjega in novega niza
    nova = s_pr + bytes([s[0]])
    # Zapisemo nov niz v PT
    PT[dict_size]=bytes(nova)
    dict_size+=1
    # Shranimo prejsnjo vrednost s
    s_pr = s
```

Če trenutnega ključa ni v slovarju PT, je potrebno slediti dodatnemu algoritmu za sporočila oblike t||z||t. Najprej pridobimo prvi znak sestavljenega sporočila (t) in poljubni niz znakov (z).

```
# Prvi znak niza (t||z||t)
t = bytes([s[0]])
```

```
# Poljubni niz znakov z
z = bytes(s[1:])
```

Na konec zadnjega dekodiranega niza pripnemo prvi znak niza t in sestavljeni niz zapišemo v seznam ST.

```
# Na konec zadnjega dekodiranega niza pripnemo prvi znak niza t
s = t + z + t
# Pripnemo niz v seznam ST
ST.append(s)
```

Shranimo zadnji niz in ga zapišemo v PT, na naslednje prosto mesto.

```
# Shrani zadnji niz
s_pr = s
# Vpisi nov niz na prazno mesto v slovar PT
PT[dict_size]= s
dict_size+=1
```

Rezultat nato sestavimo iz vseh bajtov v seznamu ST.

Ujemanje smo preizkusili z uporabo funkcije iz knjižnice hashlib in ukazom md5sum v ukazni vrstici, ki poda hash vrednost datoteke. Če v datoteki spremenimo 1 sam znak, se hash vrednost popolnoma spremeni.

```
(faks-env) spehj@spehj-thinkpad:~/faks/ik/vaja3$ python vaja3.py besedilo.txt results.txt
Rezultat dekodirnika za vhodno datoteko: besedilo.txt
MD5 vhodna datoteka:    727c3bd61031d70c31549a0ca27eb876
MD5 izhodna datoteka:   727c3bd61031d70c31549a0ca27eb876
Isti MD5 izvlecek:      True
```

Slika: Uporaba funkcije md5 iz knjižnice hashlib.

```
(faks-env) spehj@spehj-thinkpad:~/faks/ik/vaja3$ md5sum besedilo.txt
727c3bd61031d70c31549a0ca27eb876  besedilo.txt
(faks-env) spehj@spehj-thinkpad:~/faks/ik/vaja3$ md5sum results.txt
727c3bd61031d70c31549a0ca27eb876  results.txt
```

Slika: Hash vrednost izvorne in dekodirane datoteke.

Iz slik je razvidno ujemanje za izbrano datoteko besedilo.txt in datoteko results.txt, ki je rezultat kodiranja in dekodiranja.

## 3. Naloga 2

V drugi nalogi smo izvedli vrednotenje uspešnosti gospodarnega kodiranja. Implementirali smo funkcijo *izracunaj_velikost*, ki za vsako desetiško vrednost iz seznama kodiranih nizov KT (kompresirano sporočilo) najprej izvede pretvorbo v binarno vrednost, nato pa izvede še prepračun za zapis v bajtih.

```
vrednost = 0
for index, i in enumerate(KT):
    # Pridobimo string binarne vrednosti za vsako stevilko v KT
    binary = bin(i)
    # Odstranimo prefix 0b
    binary = binary[2:]
    # Sestejemo stevilo bajtov za zapis binarne vrednosti
    vrednost += int(ceil(len(binary)/8))
```

1. Preizkus gospodarnosti kodiranja na različnih vrstah datotek:
Preizkus na datoteki besedilo iz 1.vaje:

- Nekompresirana (besedilo.txt):
  - Velikost kodiranega sporocila: 459189 bajtov
  - Velikost originalnega sporocila: 1038950 bajtov
  - Razmerje izvorna/kompresirana datoteka: 2.2626
  - Razmerje kompresirana/izvorna datoteka: 0.4420
- Brezizgubno kompresirana (besedilo.zip):
  - Velikost kodiranega sporocila: 471980 bajtov
  - Velikost originalnega sporocila: 428490 bajtov
  - Razmerje izvorna/kompresirana datoteka: 0.9079
  - Razmerje kompresirana/izvorna datoteka: 1.1015

Iz rezultata je razvidno dobro kompresiranje originalne besedilne datoteke (manjša velikost). Pri kompresirani zip datoteki, pa je rezultat celo večja datoteka, kar pomeni, da dodatno kompresiranje po metodi LZW ni smiselno.

Preizkus na datoteki slika iz 1. vaje:

- Nekompresirana (slika.bmp):
  - Velikost kodiranega sporocila: 1701742 bajtov
  - Velikost originalnega sporocila: 1839122 bajtov
  - Razmerje izvorna/kompresirana datoteka: 1.0807
  - Razmerje kompresirana/izvorna datoteka: 0.9253
- Izgubno kompresirana (slika.jpeg):
  - Velikost kodiranega sporocila: 136700 bajtov
  - Velikost originalnega sporocila: 135733 bajtov
  - Razmerje izvorna/kompresirana datoteka: 0.9929
  - Razmerje kompresirana/izvorna datoteka: 1.0071

- Brezizgubno kompresirana (slika.png):
  - Velikost kodiranega sporocila: 822849 bajtov
  - Velikost originalnega sporocila: 730551 bajtov
  - Razmerje izvorna/kompresirana datoteka: 0.8878
  - Razmerje kompresirana/izvorna datoteka: 1.1263

Iz rezultatov je razvidno, da je kompresiranje po metodi LWZ uspešno v primeru nekompresirane slike, vendar tudi tu uspešnost ni veliko boljša od nekompresirane slike. Pri izgubno in brezizgubno kodiranih datotekah pa je kodirano sporočilo celo večje od originalne datoteke. Iz teh rezultatov je razvidno, da izgubno in brezizgubno kodiranih datotek ni smiselno dodatno kompresirati.

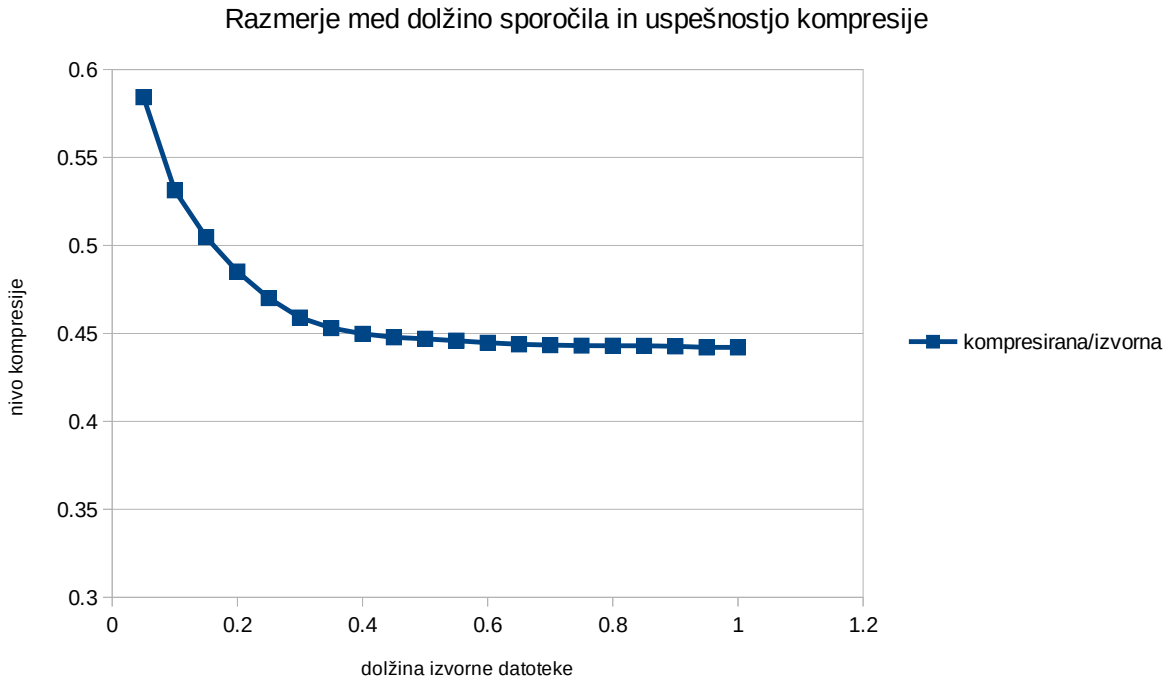
2. Preizkus gospodarnosti kodiranja pri delnem kodiranju nekompresirane datoteke:

Datoteki besedilo.txt smo v več iteracijah povečevali velikost od 0,05 MB do končne velikosti s korakom 50 00 bajtov. Rezultat shranimo v xls datoteko in dobimo naslednje vrednosti.

| datoteka [MB] | sporocilo [MB] | izvorna/ kompresirana | kompresirana/ izvorna |
|---|---|---|---|
| 0.05 | 0.03 | 1.7114 | 0.5843 |
| 0.1 | 0.05 | 1.8821 | 0.5313 |
| 0.15 | 0.08 | 1.9813 | 0.5047 |
| 0.2 | 0.10 | 2.0612 | 0.4852 |
| 0.25 | 0.12 | 2.1271 | 0.4701 |
| 0.3 | 0.14 | 2.1786 | 0.4590 |
| 0.35 | 0.16 | 2.2073 | 0.4530 |
| 0.4 | 0.18 | 2.2232 | 0.4498 |
| 0.45 | 0.20 | 2.2333 | 0.4478 |
| 0.5 | 0.22 | 2.2374 | 0.4470 |
| 0.55 | 0.25 | 2.2428 | 0.4459 |
| 0.6 | 0.27 | 2.2487 | 0.4447 |
| 0.65 | 0.29 | 2.2533 | 0.4438 |
| 0.7 | 0.31 | 2.2556 | 0.4433 |
| 0.75 | 0.33 | 2.2570 | 0.4431 |
| 0.8 | 0.35 | 2.2575 | 0.4430 |
| 0.85 | 0.38 | 2.2575 | 0.4430 |
| 0.9 | 0.40 | 2.2591 | 0.4426 |

| | | | |
|---|---|---|---|
| 0.95 | 0.42 | 2.2621 | 0.4421 |
| 1 | 0.44 | 2.2622 | 0.4420 |

Grafično prikazan rezultat:




Iz preizkusa je razvidno, da je nivo kompresije najvišji pri krajših besedilih, nato pa se z dolžino ustali v okolici neke končne vrednosti. Ugotovil sem, da je kompresiranje po metodi LZW najbolj smiselno v primeru tekstovnih datotek, saj se nizi večkrat ponavljajo in zato dodatno besedilo ne prinaša nove informacije. Pri slikah se je izkazalo, da metoda LZW ni najbolj smiselna.