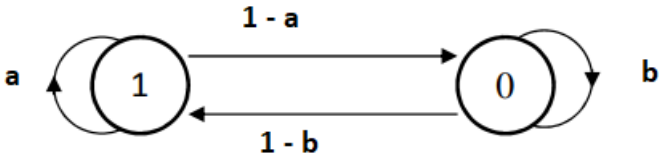
Domaći zadatak

Postavka problema:



Na slici je dat binarni izvor. Verovatnoće a I b zadaje korisnik. Za dati binarni izvor sa memorijom prvog reda odrediti : entropiju, verovatnoće pojavljivanja P(0) I P(1) , stepen kompresije I efikasnost pre same primene Hafmanovog algoritma. Nakon toga generisati txt fajl koji će oponašati rad datog sistema. Zatim primeniti Hafmanov algoritam redom na 2,3,4 I 5 proširenje izvora I za svaku primenu odrediti entropiju izvora, efikasnost , stepen kompresije I primeniti sam algoritam za kompresiju generisanog txt fajla. Na kraju predstaviti grafikom zavisnost efikasnosti od proširenja I zavisnost stepena kompresije od proširenja.

Programsko rešenje:

Programsko rešenje se sastoji od tri klase Test.Java, BinarySource.Java I HuffmanCoding.Java .

Klasa Test.Java je klasa koja služi kao interfejs sa korisnikom. On u njoj zadaje parameter binarnog izvora a I b, zadaje parametar *maximumExpansion* kojim se određuje koje je maksimalno proširenje do kog će se primenjivati Hafmanov kod I dalje preko metoda inicijalizuje *matricu verovatnoća* (*probMatrix*, objašnjena dalje u tekstu), a takođe stvara i originalni fajl koji oponaša rad sistema. Kada je izvor kreiran korisnik kreira I primenjuje Hafmana onoliko puta koliko je zadat parametar maksimalnog proširenja.

Klasa BinrySource.Java prilikom svog stvaranja redom poziva :

- *void generateProbs()* funkcija koja na osnovu zadatih parametara a I b formira verovatnoće P(0) I P(1). Za detaljno uputstvo formiranja ovih verovanoća videti [1].
- *void generateEntropy()* funkcija koja računa entropiju binarnog izvora sa memorijom prvog reda. Za detaljno uputstvo računanja entropije za ovakav izvor videti [1].
- *void generateOriginalFile(String s)* funkcija se poziva Iz klase Test.Java I za zadato ime fajla (String s) stvara originalni txt fajl koji oponaša rad našeg izvora.
- *void initializeMatrix(int p)* funkcija koja inicijalizuje matricu verovatnoće na sledeći način: svaki red matrice govori koje je proširenje u pitanju, a svaka kolona govori koja je sekvenca znakova u pitanju, a sam presek kolone I reda kolika je verovatnoća pojavljivanja date sekvence. Tako na primer *probMatrix[2][3]* nam kaže da se radi o 2+1=3 proširenju, a sekvenca simbola je 3=011.
  Naglašavam da su sve verovatnoće osim početnih(P(0), P(1), P(0|0), P(1|0), P(0|1), P(1|1))

dobijene na sledeći način: na primer verovatnoća sekvence xyz se prema Bajesovom pravilu računa kao P(xyz)=P(xy)P(z|xy) , ali kako se ovde radi o sistemu sa memorijom prvog reda kao posledica Markovljevih lanaca (videti vežbe 2.) verovatnoća P(xyz) se računa P(xyz)=P(x)P(y|x)P(z|y).

Klasa HuffmanCoding.Java preko svojih metoda za zadati binarni izvor I zadato proširenje prilikom stvaranja samog objekta, formira kodno stablo, uparuje sekvence i kodove, računa podatke za dato proširenje i generiše kompresovani txt fajl.

NAPOMENA: Iako nije od suštinskog značaja za sam domaći zadatak naglasio bih bar dve važne karakteristike koje mogu poboljšati efikasnost samog programa.

1. Matrica verovatnoća ima lepo pravilo da svaki red matrice ima tačno 2^(row+1) elemenata. Dakle radi se o *retkoj matrici* koja se može elegantno predstaviti nizom sa izmenjenim funkcijama pristupa elementima, ali uz zadržavanje trenutnog pristupa elementu. Potencijalni procenat memorijske uštede je velik(zavisno od maksimalnog proširenja).
2. Takodje iako je rekurzivno čitanje kodnog stabla elegantno takva rekurzija se može zameniti iterativnim oblisakom stabla u nekom poretku (na primer PREORDER obilazak). Ovde je takođe memorijska ušteda kao i ušteda u vremenu zbog izbegavanja višestrukog pozivanja potprograma.

Testiranje programa i rezultati:

1. Test :
   Binary source:
   a= 0.0100 , b= 0.0100
   P(0)= 0.5000 , P(1)= 0.5000
   H(S)= 0.0808
   The compression before extension: 1, the efficiency : 8.08%

   File created : original.txt
   File : original.txt has been written

   ++++++++++++++++++Initialization completed++++++++++++++++++

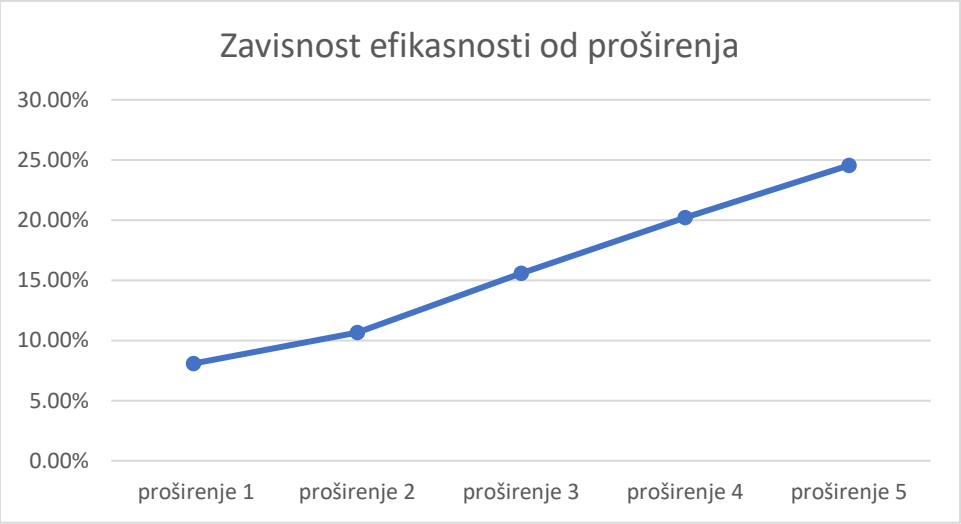
   --------The data after 2. expansion--------
   H(S): 0.1616
   Lsr: 1.5150
   efficiency: 10.6658%
   compression coef: 1.32
   Compressed file has been created

   --------The data after 3. expansion--------
   H(S): 0.2424
   Lsr: 1.5549
   efficiency: 15.5881%
   compression coef: 1.93
   Compressed file has been created

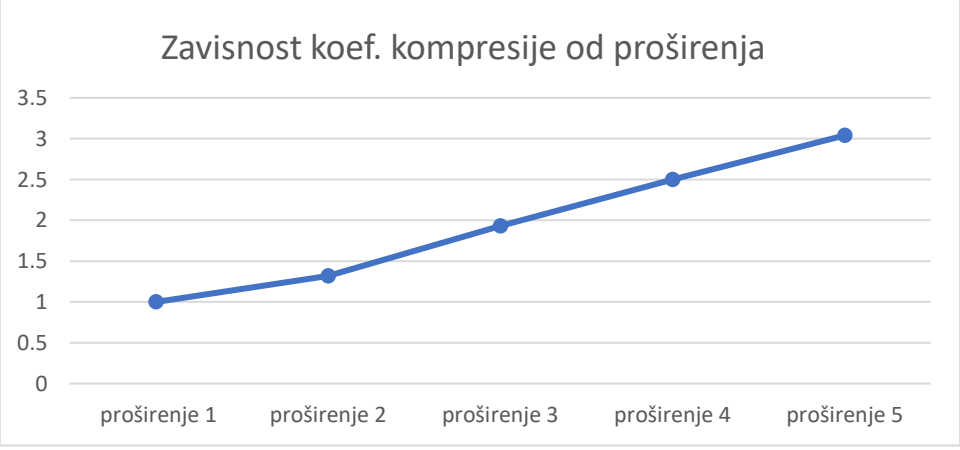
--------The data after 4. expansion--------
H(S): 0.3232
Lsr: 1.5999
efficiency: 20.1996%
compression coef: 2.50
Compressed file has been created

--------The data after 5. expansion--------
H(S): 0.4040
Lsr: 1.6456
efficiency: 24.5484%
compression coef: 3.04
Compressed file has been created

Za zadate parametre sistema dobijamo sekvencu poprilično blisku determinističkoj tj. U 99% slučajeva se nakon 0 pojavi 1 i obrnuto. Ovde je izražena memorija, a sama sekvenca nije toliko predvidiva pa je početna entropija svakako mala. Očekujemo da će već na petom proširenju ušteda biti nezanemarljiva tj. stepen kompresije će biti veći.
Primetimo da u izlistavanju rezultata programa stepen kompresije se može potvrditi tako što veličinu originalnog fajla podelimo sa veličinom kompresovanog fajla(na primer za peto proširenje stepen kompresije : 1,049,601B / 345,408B = 3,0387 tj 3,04).

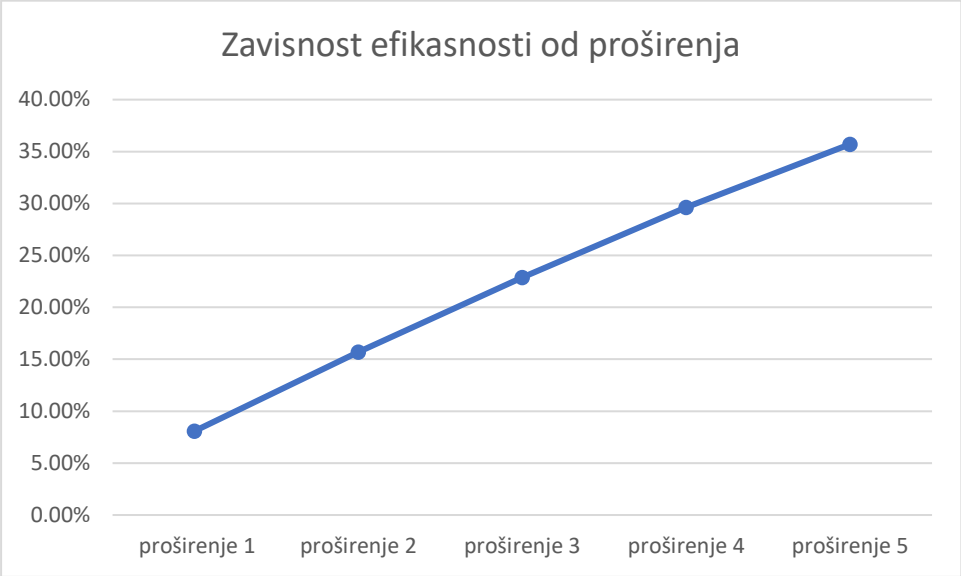


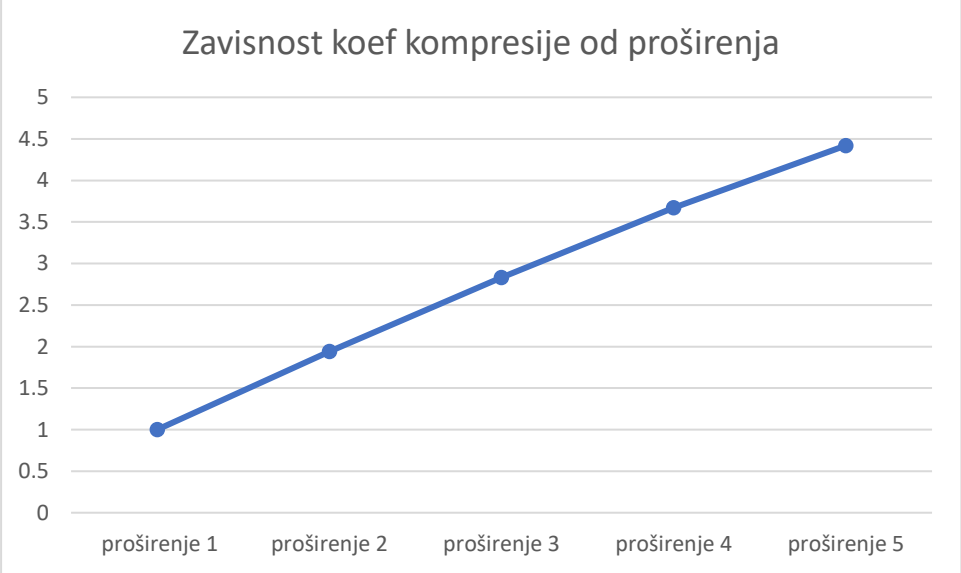
Primetimo da ni sa petim proširenjem ne dolazi do zasićenja na oba grafika.



2. Test :

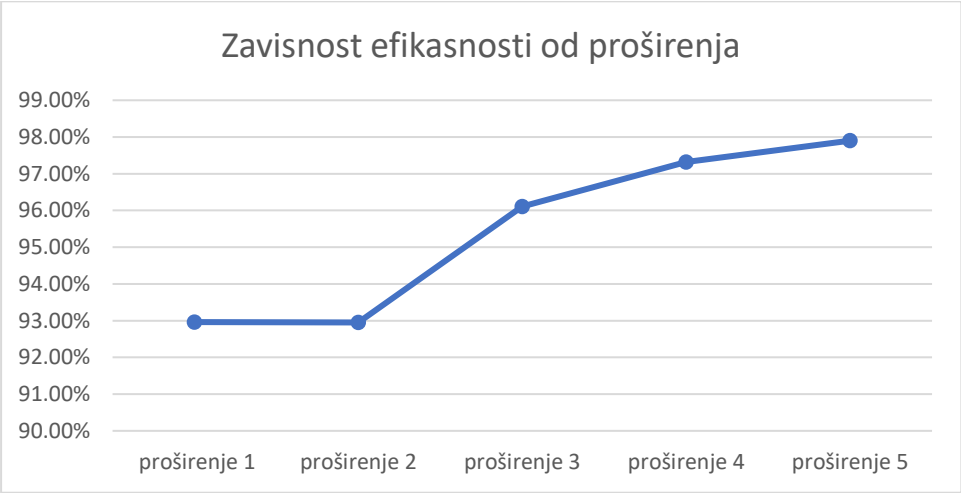
Radi preglednosti ovde ću samo navesti zadate parametre, a ne i samo izlistavanje programa. U drugom testu parametri su a=0,99 i b=0,01. Potputno neuskladjena sekvenca gde se u retkim situacijama javlja 0 a većinom su 1. Takodje je bliska determinističkoj tj. skoro da ne nosi nikakvu informaciju. Očekujemo veliku kompresiju ovakvog fajla a rezultati su sledeći:



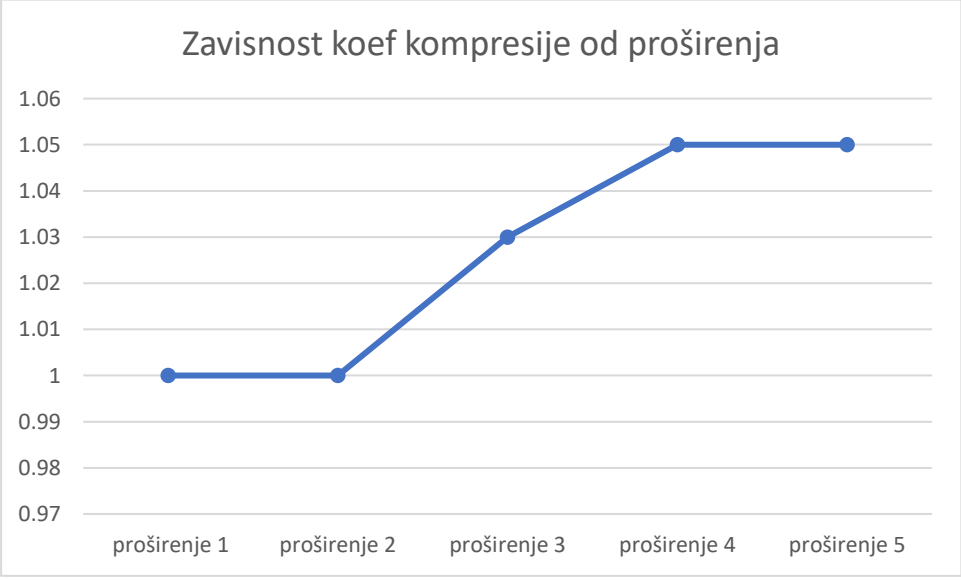


3. Test :

Treći primer je interesantan jer za unete vrednosti parametara a=0.3 i b=0.4 približavamo se sistemu sa jednako verovatnim simbolima tj. „ubijamo“ memoriju, a samim tim raste entropija, a takodje i Huffman postiže dobre rezultate jer sam algoritam ne podrazumeva memoriju već se ona veštački podrazumeva kroz sama proširenja.
Rezultati su sledeći:



Primetimo da se u drugom proširenju desio mali pad u odnosu na proširenje1 (to jest sam original), a to ne mora nužno značiti grešku jer I Šenonova teorema govori da sa proširenjem konvergiramo do 100% , ali ne kaže da konvergiramo monotono.



Koeficijent kompresije ovde nije visok iako efikasnost jeste jer upravo je to posledica „skoro“ jednako verovatnih simbola.

Za svaki test koristeći WinRar dobijeni su sledeći koeficijenti kompresije(koef kompresije dobijen deljenjem veličine originalnog fajla i fajla nakon kompresije pomoću programa) :
- za 1. test 57,4
- za 2. test 54,6
- za 3. test 6,5

Ovo mi govori da je „skoro“ determinističke sekvence kompresovao izutetno efikasno zbog izraženog prisustva memorije (1. i 2.) . Za očekivati je da je treći test daleko lošiji od prva dva zbog smanjenog prisustva memorije tj „skoro“ jednako verovatnih pojava simbola.
Postavlja se pitanje zašto su ovi koeficijenti daleko bolji od mojih. Sam kod WinRar-a je komercijalan pa se ne može videti. Postoje dva pravca: ili je moj metod pogrešan ili WinRar osim Huffman-a primenjuje i još neke algoritme za kompresiju. Iako su ovo 0 i 1, one su u txt fajlu predstavljene kao karakteri, samim tim pogodan algoritam pre Huffman-ovog bi bio LZW algoritam koji računa na postojanje memorije i pogodan je za txt fajlove, a zatim primena samog Huffman-a na dobijenu sekvencu.

Literatura :

[1] P. Ivaniš, V. Blagojević , „Uvod u digitalne telekomunikacije“ , Akademska misao, 2020. , Beograd.

Student: Uroš Marković

Br. Indeksa: 2018/0027

Odsek: Računarska tehnika i informatika