Aplikacija za Huffmanovo kodiranje

Pleš, Manuel

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA HUFFMANOVO KODIRANJE

Završni rad

Manuel Pleš

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 01.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Manuel Pleš
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 3875, 24.09.2019.
OIB studenta:	02530586369
Mentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Aplikacija za Huffmanovo kodiranje
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	01.09.2020.
Datum potvrde ocjene Odbora:	09.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.09.2020.

Ime i prezime studenta:	Manuel Pleš
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 3875, 24.09.2019.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za Huffmanovo kodiranje**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Algoritmi za sažimanje podataka bez gubitaka su procesi koji izvorne podatke predstavljaju u puno kraćem obliku pobrinuvši se pritom da je taj kraći oblik u konačnosti jednak izvornom. Danas ima mnogo različitih algoritama za sažimanje podataka bez gubitaka, no u ovom završnom radu nastoji se izraditi iOS mobilna aplikacija koja koristi samo jedan od njih. Radi se o Huffmanovom algoritmu sažimanja podataka koji se kroz rad pojašnjava, a zatim se implementira pomoću programskog jezika Swift tvrtke Apple.

Problem koji je ovim završnim radom riješen je pretvaranje koraka Huffmanovog algoritma u njihove ekvivalente u kodu bez korištenja pokazivača u modernom programskom jeziku „Swift“. Problem se razrađuje u dva dijela od kojih prvi dio predstavlja implementaciju algoritma koji je zadužen za izradu stabla, a drugi dio implementaciju algoritma koji čita rješenja iz dobivenog stabla.

Implementirani algoritmi se zatim stavljaju u funkciju pomoću jednostavnog korisničkog sučelja te se funkcionalnost aplikacije ispituje na simulatoru razvojnog okruženja „Xcode“.

U prvom poglavlju predstavljen je problem ovog završnog rada, a time i zadatak rješavanja istog. U drugom poglavlju se navode i ukratko opisuju alati koji su korišteni za izradu aplikacije za Huffmanovo kodiranje. Nakon toga se u trećem poglavlju pobliže upoznaje s teorijskom podlogom Huffmanovog algoritma potrebnog za rješavanje problema te se prikazuje primjer riješenog zadatka s objašnjenjima. Četvrto poglavlje prikazuje proces izrade aplikacije u kojem se nalaze rješenja danog problema. U petom poglavlju se opisuje izgled aplikacije i ispituje njena funkcionalnost. U šestom odnosno zadnjem poglavlju se prikazuje doneseni zaključak i iznosi vlastito mišljenje o postignutim rješenjima.

1.1. Zadatak završnog rada

Napraviti aplikaciju koja će omogućiti unos određenog broja znakova i njihovih frekvencija pojavljivanja. Od tih frekvencija generirati Huffmanovo stablo i omogućiti korisniku kodiranje i dekodiranje poruka koristeći takvo stablo.

2. KORIŠTENE TEHNOLOGIJE

2.1. Programski jezik Swift

Swift je višenamjenski programski jezik kojeg je razvila tvrtka Apple Inc. za potrebe vlastitih uređaja. Predstavljen je 2014. godine na *Worldwide Developers Conference* događaju tvrtke Apple Inc. [1]. Osmišljen je da radi s Apple-ovim Cocoa i Cocoa Touch razvojnim okruženjem te s već postojećim Objective-C kodom. Prilikom izrade programskog jezika Swift, Apple je promatrao neke od popularnih programskih jezika kao što su Objectice-C, Ruby, C#, Python i mnoge druge te nastojao iskoristiti ono najbolje od svih u Swift-u. Swift je stoga moderan, siguran i intuitivan objektno orijentiran programski jezik koji teži ka jednostavnijoj sintaksi te nastoji olakšati posao programeru. Na slici 2.1 je prikazano kako izgleda sintaksa u Swift-u te se vidi način vršenja deklaracije i inicijalizacije varijable, zajedno s pisanjem i pozivom funkcije. Najnovija verzija Swift-a trenutno je Swift 5.1 i ona je korištena u ovom završnom radu.

```
1  import UIKit
2
3  var example: String
4  // This is an example on how to declare a variable of type String in Swift.
5
6  example = "Assigning a value to a variable"
7
8  var initialization : String = "initialization"
9
10 func exampleFunction() {
11     print("We covered: \(example) and \(initialization) in Swift.")
12     // This is how you write a function in Swift
13 }
14
15 exampleFunction() // This is how you call a function in Swift.
```

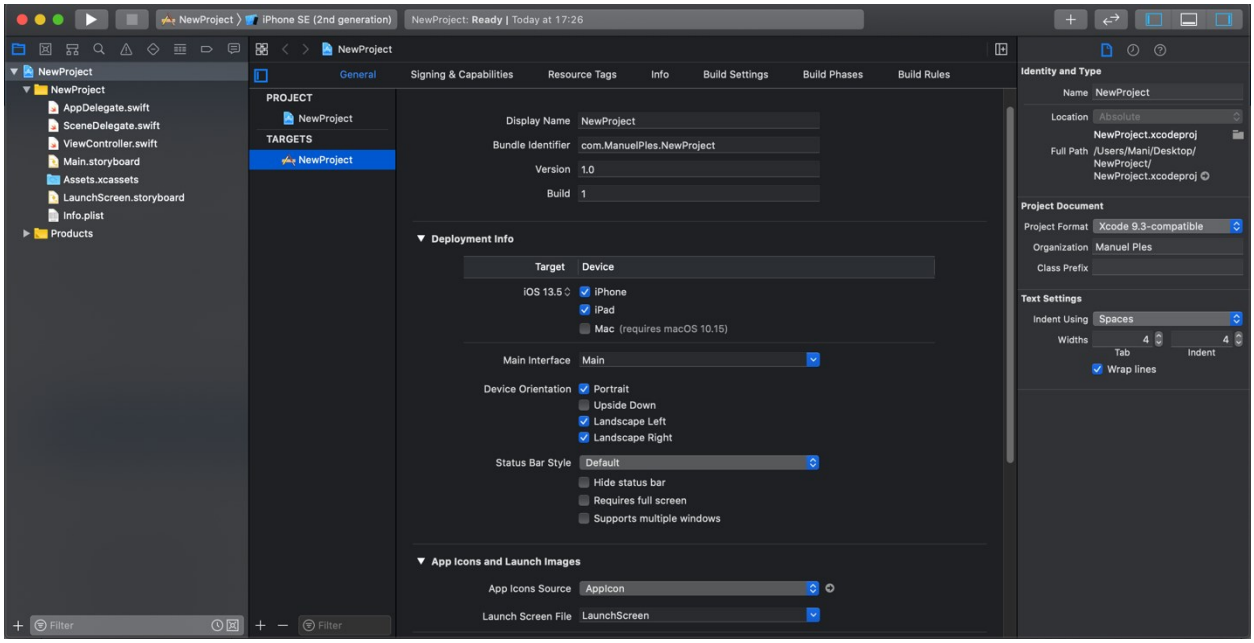
```
We covered: Assigning a value to a variable and initialization in Swift.
```

Slika 2.1. Izgled sintakse u Swift-u

2.2. Xcode IDE

Xcode je integrirano razvojno okruženje napravljeno od strane tvrtke Apple Inc. za razvoj softvera za njihove uređaje te se koristi kao jedini službeni alat za izradu i objavljivanje aplikacija na App Store[2]. Xcode je dostupan samo na uređajima koji imaju macOS i podržava veliki broj programskih jezika kao što su Objective-C, Objective C++, C, Python, ResEdit, Ruby, AppleScript, Java i Swift. On koristi Carbon, Cocoa i Java programske modele

[2]. Uz osnovne alate kao što su prevoditelj i uređivač teksta, Xcode sadrži i veliki broj alata koje koji korisniku poboljšavaju i olakšavaju pisanje programskog koda. Jedan od njegovih najkorisnijih alata je kontrolor izvornog koda koji traži greške u kodu dok korisnik unosi kod, označava ih i nudi prijedloge mogućih ispravaka. Tu je i *Auto Layout* koji uvelike olakšava stvaranje aplikacije koja odgovara svim veličinama zaslona i automatsko dovršavanje koje ubrzava proces pisanja koda. Trenutna verzija Xcode-a je Xcode 11.5 i ista je korištena u ovom završnom radu.



Slika 2.2. Izgled sučelja Xcode nakon stvaranja novog projekta

2.3.CocoaPods

CocoaPods je upravitelj ovisnosti na razini aplikacije za Swift i bilo koji drugi programski jezik koji koristi Objective-C. CocoaPods pruža standardni format za rukovanje vanjskim bibliotekama te mu je cilj omogućiti automatsko integriranje vanjskog koda u Xcode projekte [3]. Njime se upravlja preko *command line*-a i koristi se za ubrizgavanje ovisnosti kao što su vanjske biblioteke u aplikaciju bez potrebe ručnog kopiranja izvornih datoteka. Osim što koristi mnoge izvore, njegov glavni repozitorij koji sadržava meta podatke za velik broj biblioteka otvorenog koda je smješten i održavan na GitHub-u. CocoaPods je nastao prema uzoru na kombinaciju Ruby projekata Bundler i RubyGems.

3. HUFFMANOVO KODIRANJE

Kako bi se bolje shvatio proces izrade aplikacije potrebno je prvo opisati što je to i čemu služi Huffmanovo kodiranje.

Huffmanovo kodiranje dobilo je naziv po američkom pioniru Davidu Albertu Huffmanu koji ga je konstruirao za vrijeme doktorskog studija na MIT-u te je 1998. godine dobio zlatnu jubilarnu nagradu za tehnološku inovaciju IEEE-ovog društva za teoriju informacija, za „izum Huffmanovog koda kompresije podataka bez gubitka minimalne dužine“ [4].

Osnovna ideja Huffmanovog kodiranja bazira se na tome da se simbolima koji se najčešće pojavljuju dodjele kodne riječi najmanje duljine. Na takav način vrši se kompresija i povećava se efikasnost koda. Huffmanov kod je neravnomjeran kod što znači da nema jednak broj elemenata u svakoj kombinaciji. Kod se čita iz stabla koje se dobije primjenom Huffmanovog algoritma. Primjena Huffmanovog algoritma i izrada stabla prikazani su slijedećim primjerom [5].

Primjer 1

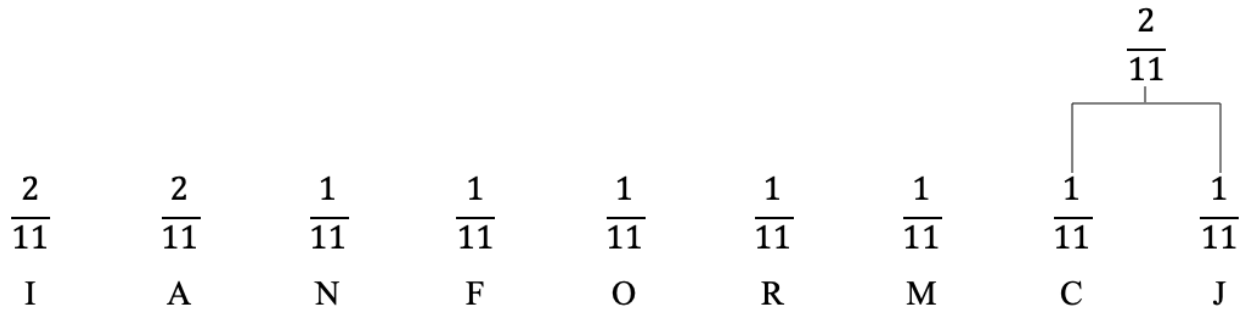
Zadatak: Iz znakova riječi „INFORMACIJA“ izračunati frekvencije pojavljivanja svakog znaka i pomoću Huffmanovog algoritma odrediti kodove za svaki znak.

1. Vjerojatnosti pojavljivanja simbola poslože se od najveće prema najmanjoj, na način da se najveća vrijednost nalazi na krajnjoj lijevoj strani, a najmanja na krajnjoj desnoj (Slika 3.1.). U ovom slučaju vjerojatnost pojavljivanja simbola je zapravo frekvencija pojavljivanja istog.

Frekvencija pojavljivanja	$\frac{2}{11}$	$\frac{2}{11}$	$\frac{1}{11}$	$\frac{1}{11}$	$\frac{1}{11}$	$\frac{1}{11}$	$\frac{1}{11}$	$\frac{1}{11}$	$\frac{1}{11}$
Simbol	I	A	N	F	O	R	M	C	J

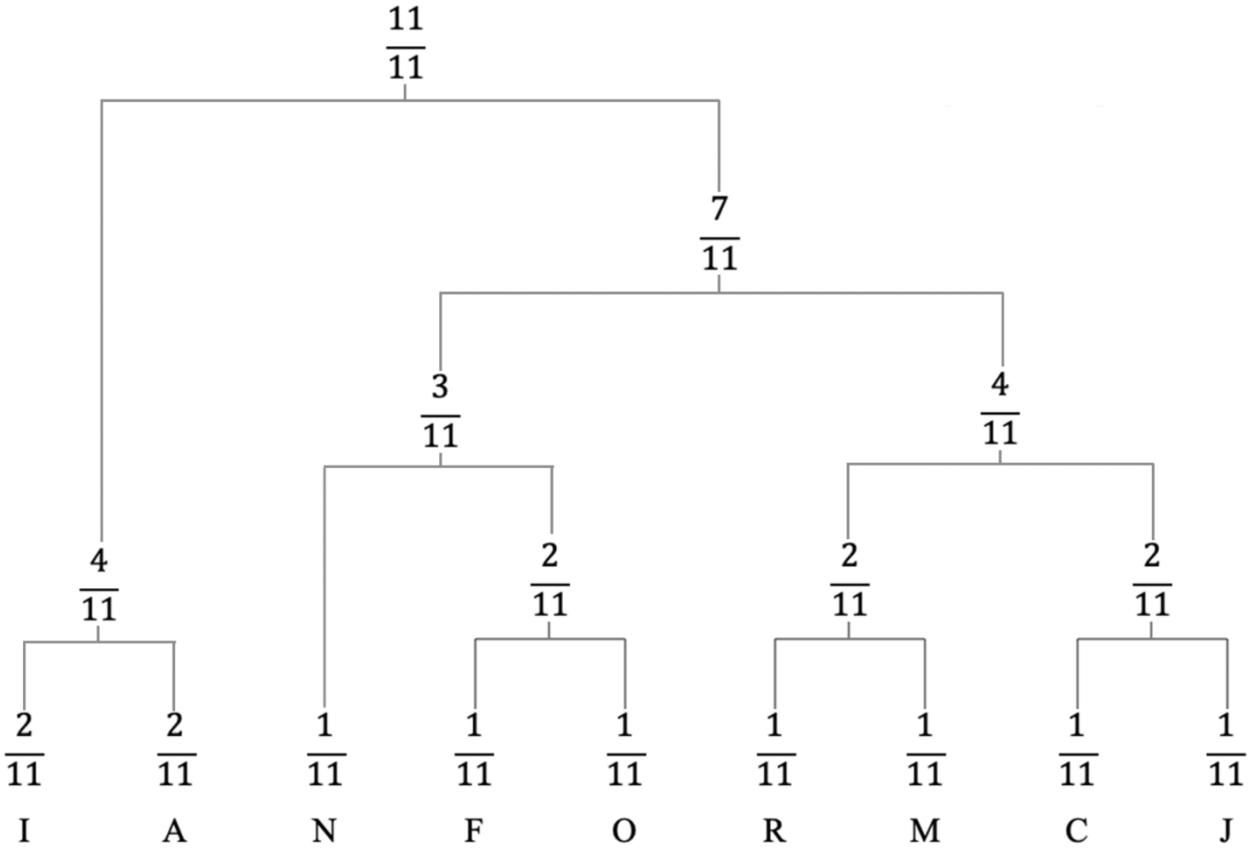
Slika 3.1. Tablica frekvencija pojave simbola

2. Dvije krajnje desne frekvencije pojavljivanja simbola se zbroje (Slika 3.2.).



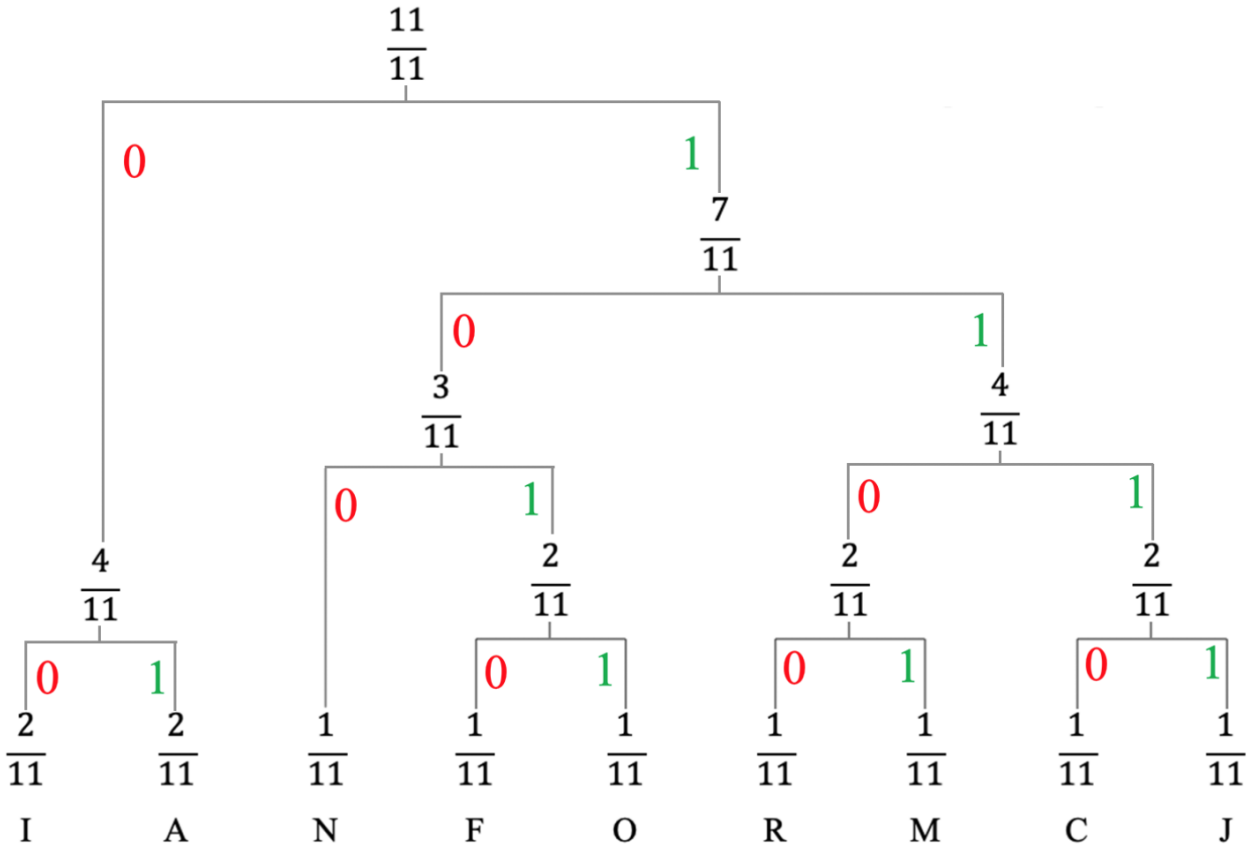
Slika 3.2. Zbroj dviju krajnjih frekvencija

3. Zbroje se dvije najmanje frekvencije pojavljivanja. Ukoliko postoji više jednakih najmanjih vrijednosti zbroje se bilo koje dvije, ali uobičajeno je da to budu dvije krajnje desne. Ovaj korak se ponavlja sve dok ne ostane samo jedna vrijednost, a ako je stablo ispravno ona će uvijek biti 1 (Slika 3.3.).



Slika 3.3. Gotovo stablo

4. Počevši od vrha prema dnu, na svakoj točki grananja dodjeljuje se po jedna binarna vrijednost (0 ili 1) svakoj grani te točke. Uobičajeno je da se lijevoj grani dodjeli vrijednost 0, a desnoj 1 (Slika 3.4.). Ukoliko se napravi obrnuto, kod će i dalje biti ispravan, ali će bit invertiran.



Slika 3.4. Stablo nakon dodjeljivanja binarne vrijednosti svakoj grani

5. Počevši od vrha, prati se najkraći put do prvog simbola i pritom se zapisuju svi bitovi na koje se nailazi. Ovaj korak se ponavlja za svaki simbol pojedinačno.

Simbol	Kod
I	00
A	01
N	100
F	1010
O	1011
R	1100
M	1101
C	1110
J	1111

Slika 3.5. Tablica s kodiranim vrijednostima

Važno je napomenuti da je Huffmanov kod zapravo kod s prefiksom što znači da prilikom dekodiranja svaku riječ možemo dekodirati na samo jedan način. Razlog tome je što niti jedna riječ ne predstavlja prefiks neke druge kodne riječi. Ukoliko to ne bi bilo tako svaka riječ bi se mogla dekodirati na više načina. Pri Huffmanovom kodiranju postoji više mogućih rješenja zbog proizvoljnog zbrajanja najmanjih vrijednosti i proizvoljnog dodjeljivanja binarnih vrijednosti granama no ta se rješenja uvijek mogu dekodirati ukoliko uz njih postoji stablo iz kojeg su iščitana.

Huffmanovo kodiranje može bit statičko ili dinamičko. Kod statičkog kodiranja koriste se iste vjerojatnosti pojavljivanja simbola i stablo se kreira samo jednom na početku kodiranja te se više ne mijenja. Ovo je ograničavajući faktor u većini primjena jer u mnogim slučajevima nije moguće odrediti vjerojatnost pojave određenih simbola ili se ona mijenja dinamički, stoga nam Huffmanovo kodiranje neće dati najveće moguće sažimanje.

Dinamičko kodiranje je kodiranje kod kojeg se prilikom svake nove uporabe vjerojatnosti pojave simbola računaju dinamički tijekom samog procesa kodiranja. Kod ovog kodiranja stablo se stalno iznova mijenja pa je s time veće vrijeme izvršavanja. U ovom završnom radu koristi se dinamičko Huffmanovo kodiranje.

4. PROCES IZRADE APLIKACIJE

S obzirom na teorijsku podlogu Huffmanovog kodiranja u trećem poglavlju ovog završnog rada, prvi i najvažniji korak u procesu izrade aplikacije je implementacija Huffmanovog algoritma za izradu stabla uz pomoć programskog jezika Swift. Prilikom implementacije nastoji se slijediti svaki korak algoritma prikazanog u Primjeru 1 iz trećeg poglavlja.

4.1. Implementacija algoritma za izradu stabla

Nakon kreiranja novog projekta u Xcode-u, za implementaciju prvog koraka algoritma napravljena je struktura *Node* (Slika 4.1.) koja predstavlja jedan čvor u stablu. Ona sadrži atribut *frequency* koji predstavlja frekvenciju pojave simbola i atribut *characters* čija će uloga postati jasna u daljnjim koracima ove implementacije. Važno je napomenuti da je atribut *characters* tipa „Any“. Tip podatka „Any“ može predstavljati instancu bilo kojeg tipa podatka i ima važnu ulogu u ostvarenju implementacije ovog algoritma [6].

```
struct Node{
    var frequency: Int = 0
    var characters: Any = ""
}
```

Slika 4.1. Struktura *Node*

Nakon kreiranja strukture *Node,* kreirana je klasa Huffman. Ona u sebi sadrži atribut *tree* koji predstavlja polje elemenata tipa *Node* i u njega se spremaju simboli koje treba kodirati te njihove pripadajuće frekvencije pojavljivanja. Uz spomenuti atribut imamo i funkciju *createHuffmanTree* koja je zadužena za kreiranje stabla. Ova funkcija implementira prvi, drugi i treći korak Huffmanovog algoritma.

Za prvi korak uzima sve elemente koje sadrži polje t*ree* i sortira ih od najvećeg ka najmanjem ovisno o njihovim frekvencijama. To je postignuto ugrađenom metodom *sort()* koja se bazira na *introsort* algoritmu sortiranja (Slika 4.2.). Introsort je algoritam sortiranja koji započinje kao quicksort i prebacuje se na heapsort kada rekurzija dostigne određenu razinu. Složenost ovog algoritma iznosi O(n log n) [7].

```
var tree: [Node]

func createHuffmanTree() {
    tree.sort{$0.frequency > $1.frequency}
```

Slika 4.2. Kreiranje i sortiranje polja

Za drugi korak algoritma u funkciji su stvorene dvije konstante, *leastFrequencyIndex1* i *leastFrequencyIndex2.* U *leastFrequencyIndex1* se pronalazi i sprema indeks elementa s najmanjom frekvencijom u polju, ali na način da ako postoji više elemenata s istom frekvencijom, uzima se indeks skroz desnog elementa. U varijablu *firstNode* se sprema element iz polja *tree* s indeksom *leastFrequencyIndex1* i zatim ga se briše iz polja. Isti postupak se ponavlja i za *leastFrequencyIndex2* i varijablu *secondNode.* Nakon toga se provjerava je li *leastFrequencyIndex1* veći od *leastFrequencyIndex2,* u slučaju da je, vrši se zamjena *firstNode* i *secondNode* kako bi se izbjeglo njihovo obrnuto umetanje u novi element new*Node* koji sadržava zbrojene frekvencije oba elementa i novonastalo polje njihovih simbola. Brisanjem elemenata promijenio se ukupni broj elemenata u polju te je potrebno postaviti uvjet koji osigurava da se novonastali element umetne na polje s postojećim indeksom. Iz tog razloga se uspoređuju *leastFrequencyIndex1* i *leastFrequencyIndex2* te se *newNode* umeće na mjesto onog manjeg u polju.

Treći korak Huffmanovog algoritma ostvaren je na način da se prethodna dva koraka stave u *While petlju* koja završava onog trenutka kada u polju ostane samo jedan element. Prilikom završavanja kreirano je stablo. (Slika 4.3.)

```
func createHuffmanTree() {
    tree.sort{$0.frequency > $1.frequency}
    while(tree.count > 1){
        var newNode = Node()
        let leastFrequencyIndex1 = tree.indices.filter{tree[$0].frequency == tree.min{$0.frequency <
            $1.frequency}?.frequency}.max()!
        var firstNode = tree[leastFrequencyIndex1]
        tree.remove(at: leastFrequencyIndex1)
        let leastFrequencyIndex2 = tree.indices.filter{tree[$0].frequency == tree.min{$0.frequency <
            $1.frequency}?.frequency}.max()!
        var secondNode = tree[leastFrequencyIndex2]
        tree.remove(at: leastFrequencyIndex2)
        if leastFrequencyIndex1 > leastFrequencyIndex2 {
            let tempNode = firstNode
            firstNode = secondNode
            secondNode = tempNode
        }
        newNode.frequency = firstNode.frequency + secondNode.frequency
        newNode.characters = [firstNode.characters, secondNode.characters]
        let newIndex = min(leastFrequencyIndex1, leastFrequencyIndex2)
        tree.insert(newNode, at: newIndex)
    }
}
```

Slika 4.3. Postupak kreiranja stabla

4.2.Implementacija algoritma za čitanje iz stabla

Nakon izrade stabla polje *tree* sadrži samo jedan element tipa *Node.* Taj element sadrži dvije vrijednosti od kojih je jedna zbroj svih frekvencija dobivenih prilikom kreiranja stabla, a druga vrijednost je ona koja sadrži podatke potrebne za kodiranje. Ti podaci se nalaze unutar atributa *characters* i imaju oblik polja u polju koje ide u dubinu sve dok ne prođe sve početne vrijednosti od kojih je stablo krenulo prije početka algoritma (Slika 4.4.).

```
[Huffman_Coding.Node(frequency: 10, characters: ["A",
    ["B", ["C", "D"]]])]
```

Slika 4.4. Izgled polja *tree* nakon primjene algoritma na niz znakova „AAAABBBCCD“

Kako bi se ostvario četvrti i peti korak Huffmanovog algoritma koji su zaduženi za prolazak kroz stablo, dodjeljivanje binarnih vrijednosti i konačno čitanje koda, u klasu *Huffman* dodana je funkcija *getEncodedData i* atribut *solution* koji će sadržavati polje simbola i njihovih pripadajućih kodova odnosno sadržavat će polje elemenata tipa prethodno kreirane strukture *CharacterCode* (Slika 4.5.). Struktura *CharacterCode* sadrži atribut *currentCode* u koji se zapisuju binarne vrijednosti prilikom prolaska kroz stablo i atribut *characters* koji ima ulogu pokazivača koji

pokazuje na trenutni *Node* u stablu. Radi lakšeg objašnjena atribut *characters* će se navoditi kao pokazivač iako nije, već samo ima njegovu ulogu.

```
struct CharacterCode{
    var currentCode = ""
    var characters: Any
}
```

Slika 4.5. Struktura *CharacterCode*

Funkcija *getEncodedData* sadrži cijeli algoritam za prolazak kroz stablo i bazira se na rekurziji. Ona radi na način da za parametar uzme element tipa *CharacterCode* koji u sebi sadrži trenutni kod elementa i pokazivač na trenutni *Node* u stablu te prvo za izlazni uvjet rekurzije provjeri pokazuje li pokazivač na samo jedan znak, ili i dalje pokazuje na polje. Ukoliko pokazivač pokazuje na samo jedan znak, trenutni element koji sadrži dosad zapisani kod i samo taj znak, dodaju se u polje *solution* i predstavlja jedno od rješenja. Ako pokazivač pokazuje na polje, stvaraju se dvije nove konstante, *newListLeft* i *newListRight* koje su tipa *CharacterCode* i imaju ulogu pokazivača od koji jedan prolazi kroz lijevu, a drugi kroz desnu stranu stabla i zapisuju odgovarajuće bitove u predani element funkcije ovisno o kojoj se grani radilo. Ovaj postupak se ponavlja sve dok se ne prođe kroz cijelo stablo i dok se svakom simbolu ne dodijeli pripadajući kod (Slika 4.6.).

```
func getEncodedData(list: CharacterCode) {
    if list.characters is Character {
        solution.append(list)
        return
    }
    let newListLeft: CharacterCode = CharacterCode(currentCode: list.currentCode + "0", characters:
        (list.characters as! [Any])[0])
    let newListRight: CharacterCode = CharacterCode(currentCode: list.currentCode + "1", characters:
        (list.characters as! [Any])[1])
    getEncodedData(list: newListLeft)
    getEncodedData(list: newListRight)
    return
}
```

Slika 4.6. Funkcija *getEncodedData()*

Kako bi se rad algoritma mogao prikazati, u algoritam su dodane linije koda koje omogućuju ispisivanje svakog koraka u konzolu (Slika 4.7.).

```
func getEncodedData(list: CharacterCode) {
    if list.characters is Character {
        solution.append(list)
        print("Caracter \(list.characters) and his code \(list.currentCode) added to solution!")
        return
    }
    let newListLeft: CharacterCode = CharacterCode(currentCode: list.currentCode + "0", characters:
        (list.characters as! [Any])[0]); print("\(newListLeft)" + " ---------LEFT----------")
    let newListRight: CharacterCode = CharacterCode(currentCode: list.currentCode + "1", characters:
        (list.characters as! [Any])[1]); print("\(newListRight)" + " --------RIGHT----------")
    getEncodedData(list: newListLeft)
    getEncodedData(list: newListRight)
    return
}
```

Slika 4.7. Proširena funkcija *getEncodedData()* koja ispisuje korake u konzolu

Na slici 4.8. vidljiv je ispis u konzoli nakon što je što je algoritam za čitanje iz stabla funkcije *getEncodedData* primijenjen na stablo nastalo primjenom algoritma funkcije c*reateHuffmanTree* na niz znakova „AAAABBBCCD“. Prva linija predstavlja rezultat funkcije *createHuffmanTree,* a ostale linije predstavljaju korake izvršavanja funkcije *getEncodedData.*

```
[Huffman_Coding.Node(frequency: 10, characters: ["A", ["B", ["C", "D"]]])]
CharacterCode(currentCode: "0", characters: "A") ---------LEFT----------
CharacterCode(currentCode: "1", characters: ["B", ["C", "D"]]) --------RIGHT----------
Caracter A and his code 0 added to solution!
CharacterCode(currentCode: "10", characters: "B") ---------LEFT----------
CharacterCode(currentCode: "11", characters: ["C", "D"]) --------RIGHT----------
Caracter B and his code 10 added to solution!
CharacterCode(currentCode: "110", characters: "C") ---------LEFT----------
CharacterCode(currentCode: "111", characters: "D") --------RIGHT----------
Caracter C and his code 110 added to solution!
Caracter D and his code 111 added to solution!
```

Slika 4.8. Koraci izvršavanja funkcije *getEncodedData()* na stablu iz slike 4.4..

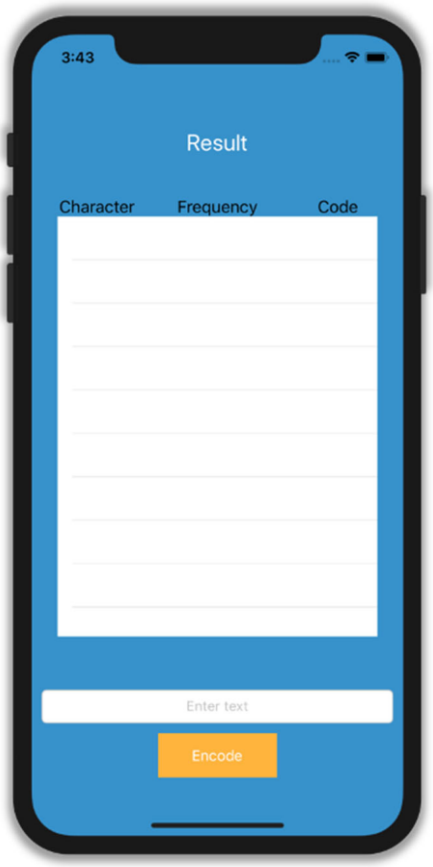
Naknadno je u klasi *Huffman* kreirana *lazy* varijabla *characterCode* tipa strukture *CharacterCode* koja sadržava gotovo stablo odnosno pokazivač na prvi element(*root*) stabla i prazan *String. characterCode* se neće kreirati prilikom kreiranja objekta klase Huffman već će se kreirati kad ju se prvi puta bude koristilo odnosno tek nakon što funkcija *createHuffmanTree* popuni stablo [8]. Ova varijabla će se nakon popunjavanja stabla kreirati u predati funkciji g*etEncodedData* kako bi se iz gotovog stabla iščitali kodovi simbola i spremili u polje *solution* (Slika 4.9.).

```
lazy var characterCode = CharacterCode(currentCode: "", characters: tree[0].characters)
```

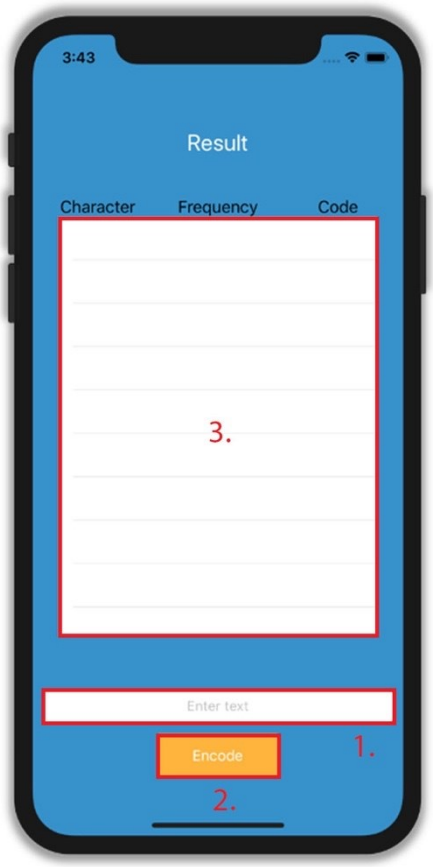
Slika 4.9. Kreiranje *lazy* varijable *characterCode*

4.3. Izrada korisničkog sučelja i povezivanje implementiranih funkcionalnosti s njim

Nakon što je cijeli Huffmanov algoritam implementiran stvoreno je korisničko sučelje koje omogućava korisniku jednostavno kodiranje unesenih simbola dodirom samo jedne tipke (Slika 4.10.).



Slika 4.10. Izgled korisničkog sučelja nakon pokretanja aplikacije



Slika 4.11. Elementi korisničkog sučelja označeni rednim brojem

Najvažniji elementi od koji se sastoji korisničko sučelje su:

1. Polje za unos teksta

Polje za unos teksta (Slika 4.11.) je u Swiftu element pod nazivom *UITextField* i koristi se kako bi korisnik mogao unijeti željeni niz znakova koji je potrebno zapisati Huffmanovim kodom. Nakon što korisnik unese željeni tekst, iduće što mora napraviti je dodirnuti *button* Encode. Dodirom polja za unos teksta otvara se tipkovnica. Kako tipkovnica ne bi smetala korisniku tako

što prekrije polje za unos teksta i *button Encode* instalirana je ovisnost IQKeyboardManager koja se brine o tome.

2. Button Encode

Button Encode (Slika 4.11.) je u Swiftu element pod nazivom *UIButton* koji se nalazi unutar glavne *ViewController* klase i nakon što ga korisnik dodirne pokreće cijeli proces Huffmanovog kodiranja (Slika 4.12.).

```
@IBAction func Encode(_ sender: UIButton) {
    clearData()
    dataTable.reloadData()
    updateResult()
        if textField.text != "" && textField.text != nil {
            userEnteredString = textField.text!
            characterFrequency = getCharacterFrequency(from: userEnteredString)
            fillTree()
            huffman.createHuffmanTree()
            huffman.getEncodedData(list: huffman.characterCode)
            sortSolution()
            dataTable.reloadData()
            updateResult()
    } else {
            textField.placeholder = "Please enter the text!"
    }
}
```

Slika 4.12. Implementacija *Encode button-a*

Proces kodiranja započinje tako što se prvo provjeri ima li u polju za unos teksta upisanih znakova. Ukoliko nema, *placeholder* tekst mijenja se iz „Enter text“ u „Please enter the text!“.

Ako polje za unos teksta nije prazno onda se tekst koji je korisnik unio sprema u varijablu *userEnteredString* koja se zatim predaje funkciji *getCharacterFrequency()* (Slika 4.13.) koja iz korisnikovog teksta stvara tablicu frekvencija pojavljivanja, odnosno broji koliko se puta koji simbol pojavio u tekstu. Ti se podaci spremaju u konstantu *characterFrequency* koja je *dictionary* to jest spremaju se kao uređeni parovi simbola i njegove frekvencije gdje je simbol ključ, a frekvencija vrijednost.

```
func getCharacterFrequency(from text: String) -> [Character : Int]{
    let characterFrequency = text.reduce([:]) { (d, c) -> Dictionary<Character,Int> in
        var d = d
        let i = d[c] ?? 0
        d[c] = i+1
        return d
    }
    return characterFrequency
```

Slika 4.13. Funkcija *getCharacterFrequency()*

Nakon što su dohvaćeni simboli i njihove frekvencije poziva se funkcija *fillTree()* (Slika 4.14.) koja uzima svaki simbol i njegovu frekvenciju i sprema ih u polje *tree* unutar klase *Huffman*.

```
func fillTree() {
    for character in characterFrequency {
        huffman.tree.append(Node(frequency: character.value, characters: character.key))
    }
}
```

Slika 4.14. Funkcija *fillTree()*

Na objektu *huffman* klase *Huffman* se zatim poziva metoda *createHuffmanTree()* koja stvara stablo na osnovu predanih simbola i njihovih frekvencija. Nakon izrade stabla na istom objektu poziva se metoda *getEncodedData()* i njoj se predaje prethodno kreirana *lazy* varijabla *characterCode* koja uzima gotovo stablo pohranjeno u polju *tree*. Metoda *getEncodedData()* prolazi kroz stablo i zapisuje sve kodove i njihove simbole u polje *solution.* Polje *solution* se zatim sortira koristeći funkciju *sortSolution()* (Slika 4.15.) kako bi prilikom prikazivanja korisniku bila posložena abecednim redom.

```
func sortSolution(){
    huffman.solution.sort{$1.characters as! Character > $0.characters as! Character}
    if characterFrequency.count == 1{
        huffman.solution[0].currentCode = "1"
    }
}
```

Slika 4.15. Funkcija *sortSolution()*

Nakon sortiranja polja *solution* vrši se poziv funkcije *updateResult()* (Slika 4.16.) koja u sebi poziva funkciju *createDictionaryFromResult()* i postavlja tekst *label-a* koji predstavlja kodiranu riječ. Funkcija *createDictionaryFromResult()* uzima rezultate dobivene čitanjem iz stabla i za svako slovo pojedinačno u varijablu *resultString* na kraj dodaje pripadajući kod.

```
func updateResult(){
    createDictionaryFromResult()
    for character in textField.text! {
            resultString.append(nodeDictionary[character] ?? "")
        }
    resultLabel.text = resultString
    }
```

Slika 4.16. Funkcija *updateResult()*

Prva stvar koja se desi dodirom *Encode*-a je poziv funkcije *clearData()* (Slika 4.17.) koja stvara novi objekt klase Huffman, briše sve prethodno unesene podatke kako prilikom novog kodiranja ne dođe do miješanja sa podacima od starog kodiranja. Zatim se poziva ugrađena metoda *reloadData()* na objektu *dataTable* koji predstavlja tablicu sa simbolima, frekvencijama pojavljivanja i pripadajućim kodovima.

```
func clearData() {
    huffman = Huffman()
    userEnteredString = ""
    resultString = ""
    nodeDictionary = [:]
    resultLabel.text = ""
    characterFrequency = [:]
}
```

Slika 4.17. Funkcija *clearData()*

3. Tablica rezultata

Tablica rezultata (Slika 4.11.) je element u Swiftu pod nazivom UITableView i koristi se za prikazivanje podataka u redovima posloženim u samo jedan stupac [9]. Tablica rezultata sadrži 3 podatka u svakom redu. Ti podaci su *Character* koji predstavlja simbol, *Frequency* koji predstavlja frekvenciju pojave tog simbola i *Code* koji predstavlja Huffmanov kod za taj simbol. Kako bi *tableView* nužno je implementirati dvije funkcije, a to su *tableView(_:numberOfRowsInSection:)* (Slika 4.18.) i *tableView(_:cellForRowAt:)* (Slika 4.19.).

Prva funkcija je bitna jer se u njoj definira broj redaka koji će tablica imati. U ovom slučaju je to onoliko koliko ima elemenata u polju *solution*.

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

    return huffman.solution.count
}
```

Slika 4.18. Funkcija *tableView(_:numberOfRowsInSection:).*

U drugoj funkciji se kreira ćelija koja predstavlja jedan redak i toj se ćeliji pridružuju svi podaci koji se u njoj moraju prikazivati.

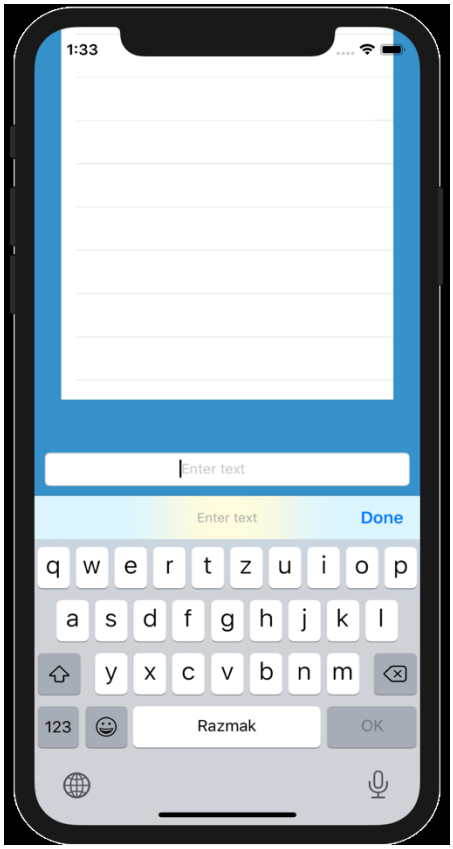
```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath) as!
        CharacterTableViewCell
    let character = huffman.solution[indexPath.row].characters as! Character
    if character != " " {
    cell.characterLabel.text = "\(character)"
    }else {
        cell.characterLabel.text = "(space)"
    }
    cell.characterFrequencyLabel.text = "\(characterFrequency[character]!)"
    cell.characterCodeLabel.text = "\(huffman.solution[indexPath.row].currentCode)"
    cell.selectionStyle = .none

    return cell
}
```

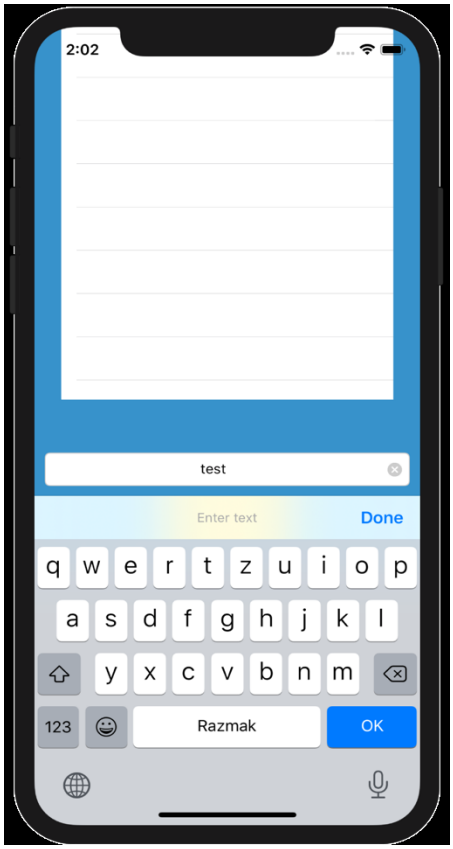
Slika 4.19. Funkcija *tableView(_:cellForRowAt:)*

5. ISPITIVANJE FUNKCIONALNOSTI APLIKACIJE

Nakon pokretanja aplikacije korisnik mora dodirnuti polje za unos teksta kako bi mogao unijeti tekst koji želi kodirati. Kada se dodirne polje za unos teksta, zbog već ugrađenih funkcija otvara se tipkovnica čiji se ulazni podaci automatski vežu za polje za unos teksta. Kako tipkovnica ne bi prekrila polje za unos teksta, instalirana vanjska ovisnost *IQKeyboardManager* pomjera ono što korisnik vidi prema gore i omogućava mu neometani upis željenog niza znakova (Slika 5.1.). Prilikom upisa teksta u polju za unos, omogućuje se *button* „OK“ u donjem desnom kutu tipkovnice i pojavljuje se *button* koji omogućuje brisanje trenutno upisanog teksta (Slika 5.2.).

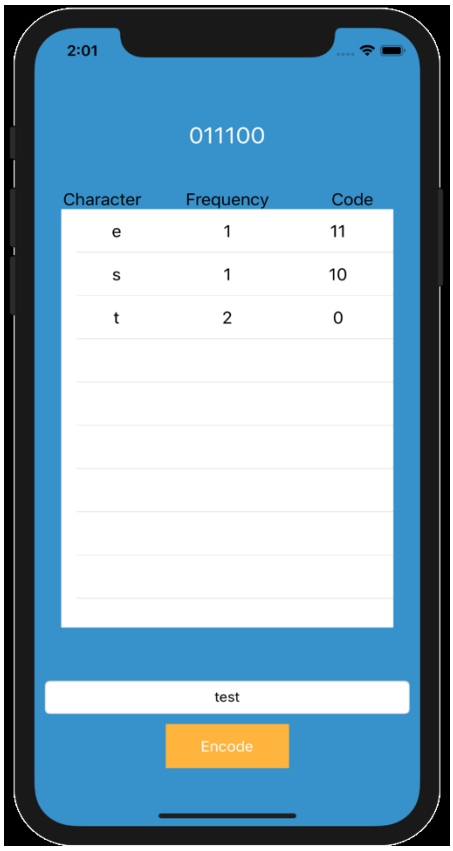


Slika 5.1. Izgled sučelja nakon dodira polja za unos teksta

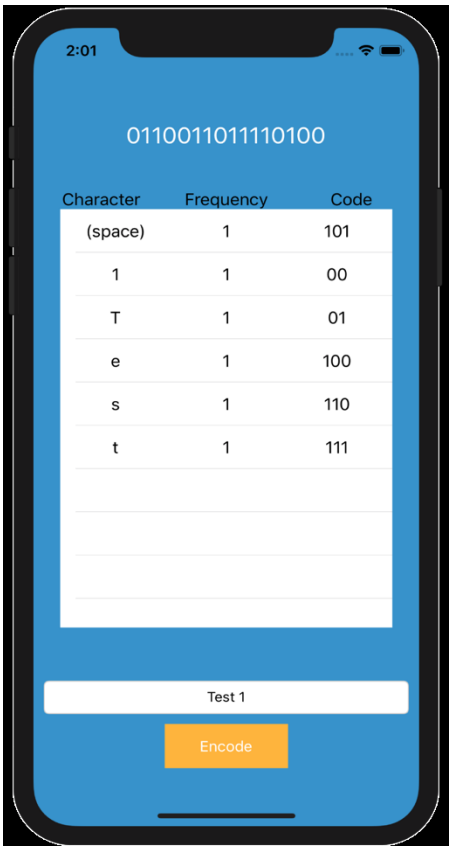


Slika 5.2. Unos teksta u polje

Nakon što je korisnik upisao tekst koji je potrebno kodirati, proces kodiranja se pokreće dodirom *button-a* „OK“ ili dodirom *button* „Done“ pa dodirom *button-a Encode.* Tipkovnica će zatim nestati i nakon što se cijeli proces u pozadini izvrši, tablica rezultata će se popuniti sa svim simbolima, njihovim frekvencijama pojavljivanja i kodovima (Slika 5.3.). Uz to će se iznad tablice rezultata ispisati kodirana riječ koju je korisnik unio u polje.



Slika 5.3. Rješenja upisana u tablicu rezultata



Slika 5.4. Prepoznavanje razmaka i razlikovanje velikih i malih slova

Na slici 5.4 je vidljivo da algoritam razlikuje velika i mala slova te da prepoznaje razmake. To je nužno kako bi se informacija nakon dekodiranja poklopila s izvornom informacijom. U ovoj aplikaciji koristi se dinamičko Huffmanovo kodiranje stoga je prisutno najveće moguće sažimanje. Zbog toga što *dictionary* nije moguće sortirati a u njega se podaci spremaju nasumično, kodovi simbola se mogu mijenjati iako je ista riječ u pitanju. Drugim riječima neće se prikazivati samo jedno rješenje već će se svakim novim pokretanjem procesa izmjenjivati sva moguća rješenja za tu riječ što je u potpunosti točno.

6. ZAKLJUČAK

U ovom završnom radu obrađen je Huffmanov algoritam za sažimanje podataka te je na primjeru prikazana njegova primjena. Prateći korake Huffmanovog algoritma, implementiran je i objašnjen algoritam za kreiranje stabla, a potom je isto napravljeno i za algoritam za čitanje iz stabla uz pomoć programskog jezika Swift.

Dobiveno rješenje uvijek pruža najveće moguće sažimanje no nije najučinkovitije iz razloga što se radi o dinamičkom Huffmanovom kodiranju koje frekvencije pojavljivanja simbola stalno iznova računa te se sa svakom novom upotrebom iznova radi stablo. Ukoliko bi se koristilo statičko Huffmanovo kodiranje kod kojeg je stablo uvijek isto odnosno koje koristi već poznate vjerojatnosti pojave simbola nekog jezika, kodiranje bi bilo puno učinkovitije, ali rješenje ne bi u većini slučajeva bilo u najsažetijem obliku. Razlog tome je što ako se uzmu dosada poznate vjerojatnosti pojavljivanja simbola u hrvatskom jeziku, one neće dati iste rezultate primjenom Huffmanovog algoritma na tekst knjige iz književnosti i na tekst knjige iz elektrotehnike u kojoj se puno više koriste nekakvi simboli.

Veliku ulogu u ostvarivanju implementacije spomenutih algoritama je imala mogućnost modernih programskih jezika da koriste tip podatka „Any“ koji je u potpunosti mogao zamijeniti ulogu pokazivača. Iako ga se ne preporučuje koristiti, u ovom završnom radu je imao jako važnu ulogu.

Funkcionalnost aplikacije je testirana i provjerena na simulatoru razvojnog okruženja Xcode i rješenje je u potpunosti ispravno.

LITERATURA

[1] Apple announces Swift, a new programming language for iOS and OSX, https://thenextweb.com/apple/2014/06/02/apple-announces-swift-new-programming-language-ios/ (pristupljeno 16.06.2020.)

[2] What is Xcode and why do I need it?, https://www.zerotoappstore.com/what-is-xcode-and-why-do-i-need-it.html (pristupljeno 16.06.2020.)

[3] CocoaPods, https://en.wikipedia.org/wiki/CocoaPods (pristupljeno 17.06.2020)

[4] David A Huffman, https://en.wikipedia.org/wiki/David_A._Huffman (pristupljeno 19.06.2020)

[5] Togneri R., deSilva Christopher J.S., Fundamentals of Information Theory and Coding Design, str. 137. Boca Raton, Florida: CRC Press. ISBN 978-0-203-99810, 2003.

[6] Type Casting for Any and AnyObject https://docs.swift.org/swift-book/LanguageGuide/TypeCasting.html (pristupljeno 20.07.2020)

[7] Introsort https://en.wikipedia.org/wiki/Introsort (pristupljeno 20.07.2020)

[8] Lazy Stored Properties https://docs.swift.org/swift-book/LanguageGuide/Properties.html (pristupljeno 21.07.2020)

[9] UITableView https://developer.apple.com/documentation/uikit/uitableview (pristupljeno 21.07.2020)

SAŽETAK

U ovom završnom radu kreirana je iOS mobilna aplikacija za Huffmanovo kodiranje. Objašnjen je Huffmanov algoritam i alati koji se koriste za izradu aplikacije. Problem s kojim se susreće je pretvaranje koraka Huffmanovog algoritma u njihov ekvivalent u kodu i rješenje koje se nudi dano je u dva dijela. Prvi dio implementacije zadužen je za izradu Huffmanovog stabla koje u programu izgleda drugačije od običnog stabla, ali ima istu funkciju. Drugi dio zadužen je za prolazak i čitanje stabla i to je postignuto rekurzivnom funkcijom. Veliku ulogu u ostvarenju rješenja ovog problema ima mogućnost programskog jezika Swift da koristi tip podatka „Any“ koji u potpunosti mijenja ulogu pokazivača. Cijeli proces sakriven je iza jednostavnog korisničkog sučelja koje omogućava korisniku upisivanje znakova koje želi kodirati i njihovo kodiranje pritiskom samo jednog dugmeta.

Ključne riječi: Huffmanov algoritam, Huffmanov kod, Huffmanovo stablo, Swift, iOS,

ABSTRACT

Application for Huffman coding

In this bachelor's thesis an iOS mobile application was created. The Huffman algorithm and the tools used to create the application were explained. The problem encountered is transforming the steps of the Huffman algorithm into their equivalent in code and the answer offered is given in two parts. The first part of the implementation is in charge of creating the Huffman tree which looks different in the program than the usual tree but has the same functionality. The second part of the implementation is in charge of going through and reading from the tree and that was accomplished with a recursive function. A great part in achieving the goal of solving this problem was the possibility of the programming language Swift to use the data type “Any” which completely replaced the role of pointers. The whole process was hidden behind a simple user interface which allows the user to enter the characters he wants to encode and then encodes them with the press of a button.

Keywords: Huffman algorithm, Huffman code, Huffman tree, Swift, iOS

ŽIVOTOPIS

Manuel Pleš rođen je 1. veljače 1997. godine u Vinkovcima. Od 2003 do 2011 godine pohađa Osnovnu Školu Ivana Gorana Kovačića u Vinkovima. Za vrijeme osnovnoškolskog obrazovanja sudjelovao je na županijskom natjecanju iz Engleskog jezika. Godine 2011. upisuje Tehničku školu Ruđera Boškovića u Vinkovcima, smjer tehničar za mehatroniku. Nakon što je završio srednju školu, 2015. godine upisao je preddiplomski studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.