Elektrotehnički fakultet
Univerziteta u Sarajevu

SEMINARSKI RAD

Vizuelizacija kompresije podataka bez gubitaka putem metoda: Shanon-Fano i Huffman

MULTIMEDIJALNI SISTEMI

Tim:

- Kenan Hadžirović
- Haris Halilović
- Dženita Ljevo
- Mirza Vučijak

Mentori:

- Red. Prof. Dr. Haris Šupić
- Kenan Ekinović,
- Orhan Ljubunčić

Sarajevo, decembar, 2018.

Opis zadatka

Algoritmi za kompresiju podataka mogu se efikasno izvršavati računarski ali i ručno. Obzirom da je ručno izvršavanje algoritma nepraktično za poruke duže od nekoliko desetina karaktera ili samo nekoliko različitih karaktera, oni se izvršavaju ručno uglavnom u edukativne svrhe. Pomoćni alati – tabele i grafovi koji se crtaju prilikom ručnog izvršavanja na papiru se ne moraju koristiti u kreiranju algoritma za kompjutersko izvršavanje. Prilagođavanje algoritama tim metodama zauzima više memorije i troši više procesorskog vremena.

Zadatak ovog seminarskog rada jer uključiti grafički prikaz podataka u algoritme koji se izvršavaju na računaru. Nakon što se poruka komprimira, potrebno je iscrtati binarno stablo kompresije koje odgovara unesenim podacima. Na osnovu tog binarnog stabla moguće je isčitati kodove svakog unesenog znaka, analizirati strukturu, kao i frekvenciju pojavljivanja znakova u unesenoj poruci. Konačno, moguće je izračunati stepen kompresije i na taj način utvrditi efikasnost kompresije.

Kao dodatna mogućnost, korisniku je omogućeno praćenje generisanja binarnog stabla kroz korake. Ti koraci uključuju prikaz prvobitnih čvorova koji su kreirani od znakova poruke. Nakon toga se čvorovi spajaju kreirajući binarno stablo odluke za kompresiju. Svakim sljedečim korakom dolazi do spajanja postojećih i dodavanja izvornog čvora, dok ne dođe do spajanja svih čvorova u jedinstveno binarno stablo. Korisnik može navigirati kroz korake naprijed ili nazad po vlastitim željama i na taj način pratiti proces generisanja.

IZJAVA O DOPRINOSU ČLANOVA TIMA

PRI IZRADI SEMINARSKOG RADA IZ PREDMETA MMS

U tabeli ispod navesti imena svih članova tima, te aktivnosti u kojima je učestvovao svaki pojedini član tima. Pri navođenju aktivnosti za svakog člana tima koristite bilo koju kombinaciju sljedećih opcija:

- Analiza problema semenarskog rada;
- Osmišljavanje rješenja;
- Praktična implementacija seminarskog rada;
- Pisanje teksta seminarskog rada;
- Ostalo (navesti koje su to ostale eventualne aktivnosti).

Prezime i ime člana tima	**Aktivnosti**
Hadžirović Kenan	– Osmišljavanje rješenja; – Praktična implementacija seminarskog rada; – Pisanje teksta seminarskog rada; – Ostalo (navesti koje su to ostale eventualne aktivnosti) a. Prijava teme i slanje seminarskog rada
Halilović Haris	– Praktična implementacija seminarskog rada; – Pisanje teksta seminarskog rada;
Ljevo Dženita	– Analiza problema semenarskog rada; – Osmišljavanje rješenja; – Praktična implementacija seminarskog rada; – Pisanje teksta seminarskog rada;
Vučijak Mirza	– Analiza problema semenarskog rada; – Osmišljavanje rješenja; – Praktična implementacija seminarskog rada;

Sadržaj

Uvod

Kompresija podataka je proces zapisivanja ili prikazivanja podataka u što kraćem obliku. Ti podaci moraju pružati mogućnost dekompresije tako da se izvorna poruka može ponovno shvatiti. Dijeli se na kompresiju sa gubicima i bez gubitaka. Kompresija bez gubitaka je proces gdje su izlazni rekonstruisani podaci jednaki ulaznim izvornim podacima. Ovaj proces se primjenjuje kod podataka kod kojih bilo kakva promjena nastala gubicima može utjecati na smisao poruke koju prenosimo. Kao primjer možemo spomenuti tekstualne poruke, gdje je jako bitno da se ne izgube nikakvi podaci jer može doći do pogrešnog tumačenja poruke. Za razliku od teksta, slika je primjer medija u kojem kompresija može sadržavati gubitke. Na primjer, ukoliko određeni pikseli slike izgube potpunu preciznost o boji koju prikazuju, dobra je šansa da i dalje možemo shvatiti poruku koju slika prenosi, čak ne primjećujući da je uopće došlo do promjene u boji ili obliku.

Fokus ovog rada je kompresija podataka bez gubitaka. Prikazane su metode kompresije podataka bez gubitaka Shanon-Fano i Huffman. Algoritmi ovih metoda su prilagođeni efikansnom izvršavanju na računarima. Međutim, kako bi se grafički prikazale, metode su dijelom modifikovane što utječe na njihovu efikasnost kako u brzini izvršavanja, tako i u zauzeću memorije. Algoritmi su podređeni grafičkom prikazivanju, te je očuvana samo njihova ideja, dok je došlo do djelomičnog odstupanja od pseudokoda koji se inače koristi za njihovu realizaciju.

Teoretska pozadina ovih algoritama bit će opisana u poglavljima koja slijede. Rad će pružiti kratka objašnjenja procesa kompresije općenito, kompresije podataka sa gubicima i bez gubitaka. Bit će prikazan pseudokod koji je poslužio kao referenca za pisanje algoritma u praktičnom dijelu.

Nakon toga, poglavlja će biti posvećena praktičnoj realizaciji projekta te prikazivanju rezultata grafičke reprezentacije Shanon-Fano i Huffmanovog algoritma kompresije. U tekst rada će biti uključeni isjećci koda koji su značajni za shvatanje internog funkcionisanja programa, kao i njihova objašnjenja. Jednostavne, pomoćne ili funkcije koje nisu ključne za funkcionalnost kompresije će biti izostavljene iz objašnjenja kako bi rad očuvao izvornu tematiku.

Kompresija podataka

Kompresija podataka je proces koji omogućava pohranu podataka u što kraćem obliku. Podrazumijeva smanjenje broja bita, te tako ubrzava prenos podataka, smanjenje prostora potrebnog za spašavanje i sl. Svaka kompresovana datoteka je zapravo „umanjena“ na veličinu s kojom je lakše manipulirati – posebno kada je prenos podataka u pitanju. Može se primijeniti na različite vrste podataka, od tekstualnih do zvučnih i video zapisa. Proces kompresije se realizuje enkoderom, dok za se dekompresiju koristi dekoder. Bilo koji tip podataka se može kompresovati, ali od konteksta je potrebno pažljivo izabrati način na koji se to radi. Prije svega treba identifikovati strukturu koju je potrebno kompresovati, a onda na osnovu toga odlučiti koju tehniku koristiti. [1][4]

Postoje dvije vrste kompresije:

- Kompresija sa gubicima
- Kompresija bez gubitaka

Kompresija sa gubicima

Kod ove vrste kompresije izlazni rekonstruirani podaci nisu jednaki ulaznim izvornim podacima. Samom primjenom ove tehnike mogu se postići veći nivoi kompresije, jer se trajno eliminiše dio bita koji su suvišni, nebitni ili neprimjetni. Podaci kompresovani ovom tehnikom se ne mogu, u potpunosti, nakon kompresije vratiti u izvorni oblik. Radi na tom principu, da ako nam neka informacija ne treba, možemo je jednostavno ukloniti, što uveliko ovisi od konteksta. Ova vrsta kompresije se najčešće upotrebljava prilikom kompresije slika i videa [1] [4] [5]. Grafički je prikazana slici 1.

Vrste podataka koji koriste kompresiju sa gubicima:

- JPEG (eng. Joint Photographic Expert Group)
- MPEG (eng. Motion Pictures Expert Group)
- MPEG (eng. Audio Layer 3)

Slika 1 - Grafički prikaz kompresije sa gubicima [6]

Kompresija bez gubitaka

Ova vrsta kompresije omogućava kompresiju podataka na način da se prilikom rekonstrukcije izvornih podataka ne pojavljuju nikakvi gubici. Koriste se u područjima primjene gdje je vrlo bitno da nema razlike između izvornih i rekonstruiranih podataka. Također se koristi kada želimo kompresovati veliki broj velikih podataka ili jednu veliku datoteku. Jedan od primjera takvih podataka su tekstualni podaci [1]. Takvi podaci su i datoteke bankovnih računa, datoteke koje sadrže programe. Ako se izgubi makar i jedan bit tog programa, on bi mogao postati neupotrebljiv [4]. Slika 2 grafički prikazuje kompresiju bez gubitaka.

Neki od najpoznatijih algoritama koji koriste ovu tehniku kompresije su:

- Shannon Fano
- Huffman

Fokus je stavljen na ove algoritme obzirom da se oni obrađuju i u praktičnom dijelu rada.

Slika 2 - Grafički prikaz kompresije bez gubitaka

Shanon-Fano metoda

Ova metoda je jedna od najstarijih kada je u pitanju kompresija podataka. Razvili su je Claude Shannon i Robert Fanno, neovisno jedan od drugog 1949.[1] Poboljšane verzije se u modernoj primjeni koriste kod kompresije podataka u .zip i .rar formate. Kodiranje se zasniva na vjerovatnoćama pojavljivanja dijelova podataka. Krajnji cilj ove metode jeste da riječi koje su najduže imaju najkraće kodove zapisa[7]. Ova metoda/algoritam ima pet osnovnih koraka:

- Korak 1: Izračunati frekvenciju pojavljivanja za svaki od simbola.
- Korak 2: Lista simbola se sortira prema frekvenciji pojavljivanja, u nerastućem poretku. Prvi treba da bude simbol koji se najviše pojavljuje a iza njega simboli koji se rijeđe pojavljuju.
- Korak 3: Lista se dijeli na dva dijela, gdje ukupna frekvencija pojavljivanja lijeve strane treba približna ukupnoj frekvenciji pojavljivanja desne strane.
- Korak 4: Lijevom dijelu liste se dodjeljuje simbol 0, dok se desnom dijelu liste dodjeljuje simbol 1, može i obrnuto. Znači da kodovi za simbole sa lijeve strane počinju 0, a sa desne strane počinju sa 1.
- Korak 5: Koristiti rekurziju prilikom primjene koraka 3 i 4 na obje strane liste. Primjenjuje se sve dok se na lijevoj ili desnoj strani ne nađe samo jedan element u listi [1]

Generalno, Shannon Fano metoda ne garantuje generisanje optimalnog rezultata, niti jednog rješenja. Ako neka dva simbola imaju istu frekvenciju, može se desiti da prilikom druge primjene ovog algoritma ta dva simbola zamijene kodove koje su imali originalno. Ovo se dešava zbog mogućnosti izbora bilo koje vrijednosti sa jednakom vjerovatnoćom pojavljivanja za naredni simbol koji je potrebno kodirati. [10]

Huffman metoda

Huffman metoda je nastala 1951. godine a razvio je David A. Huffman. Za razliku od Shannon Fanoa, gdje se stablo kodiranja izgrađuje od vrha ka dnu, ova metoda stablo kodiranja izgrađuje od dna ka vrhu. Koristi tabelu frekvencija pojavljivanja svakog simbola, kako bi na najbolji način prezentovala svaki simbol preko koda. Kompleksnost ove metode je O(nlogn) [8] [1].

Metoda/algoritam ima pet osnovnih koraka:

- Korak 1: Izračunati frekvenciju pojavljivanja za svaki od simbola.
- Korak 2: Sortirati listu prema frekvenciji pojavljivanja u opadajućem poretku. Simbol koji se najčešće pojavljuje je prvi a iza njega simboli koji se rijeđe pojavljuju.
- Korak 3: Ponavljati sljedeće korake, sve dok u listama ne bude samo jedan element
    a) Iz liste uzeti dva simbola sa najnižom frekvencijom pojavljivanja, te kreirati podstablo, tako da se generiše roditeljski čvor koji kao čvorove djecu ima ova dva simbola
    b) Novokreiranom roditeljskom čvoru pridružiti sumu frekvencija njegove djece, te taj roditeljski čvor umetnuti u listu na način da se ne mijenja poredak frekvencija pojavljivanja
    c) Obrisati čvorove djecu iz liste
- Korak 4: Svim granama u izgrađenom stablu kodiranja koje spajaju roditeljske čvorove i lijeva podstabla, treba pridružiti bit 0. Nadalje, svim granama koje spajaju roditeljske čvorove i desna podstabla, treba pridružiti bit 1.
- Korak 5: Pridružiti odgovarajuće kodne riječi za svaki list u izgrađenom stablu kodiranja. Kodne riječi se dobivaju tako da se formira niz bitova, koji se dobije tako da se uzmu pridružene oznake grana na putu od korijena do dotičnog lista [1]

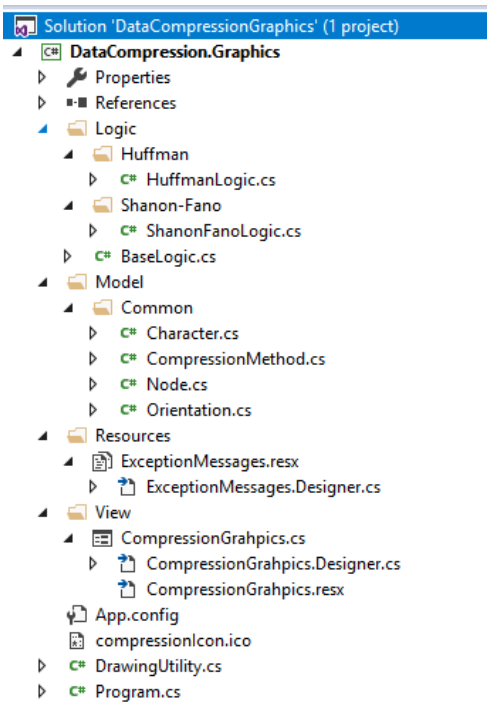
Huffmanovo kodiranje posjeduje dvije bitne osobine:

- svojstvo jedinstvenog prefiksa
- svojstvo optimalnosti

Svojstvo jedinstvenog prefiksa znači da nijedna kodna riječ dobivena ovim kodiranjem nije prefiks bilo koje druge riječi. Svojstvo optimalnosti znači da kodne riječi za dva simbola sa najmanjom frekvencijom ponavljanja imaju istu dužinu i razlikuju se po bitu koji se nalazi na posljednjoj poziciji, simboli koji imaju veće frekvencije pojavljivanja imaju kraće Huffmanove kodove od simbola sa manjim frekvencijama pojavljivanja [1].

Praktična realizacija projekta

Cilj projekta je grafički prikazati binarna stabla odlučivanja kreirana prilikom procesa kompresije podataka metodama Shanon-Fano i Huffman. Kako bi se to postiglo, korišten je .NET Framework 4.5 uz C# programski jezik unutar Visual Studio programskog okruženja. Prilikom dizajniranja arhitekture projekta, posvećena je posebna pažnja zajedničkim karakteristikama algoritama, kako bi se kod pisao poštujući principe objektno orjentisanog programiranja. Korišteni su principi naslijeđivanja, podjele odgovornosti kao i zaštite (enkapsulacije) podataka. Na narednoj slici je prikazana arhitektura koje je nastala prilikom faze dizajniranja. Programsko rješenje je podijeljeno u slojeve poštujući „Layer“ arhitekturu razdvajajući logiku, modele i prikaz u zasebne dijelove.



Slika 3 - Prikaz slojeva aplikacije

Na slici je moguće uočiti slojeve:

- **Model** – predstavlja sloj klasa za spašavanje podataka. Objekti tipva Character, CompressionMethod, Node i Orientation dovoljni su za spašavanje svih podataka unutar programa. Najvažniji je model **Node** oko kojeg se gradi sva funkcionalnost spašavanja čvora, kao I njegovog grafičkog prikazivanja
- **Logic** – predstavlja sloj biznis logike, koji u slučaju ovog programskog rješenja čine algoritmi za kompresiju. Bazna klasa BaseLogic implementira funkcionalnosti koje su

zajedničke za obje metode kompresije, dok klase ShanonFanoLogic i HuffmanLogic, pored toga što nasljeđuju BaseLogic, dodaju vlastite implementacije putem metoda Compress

- **View** – predstavlja sloj zadužen za prikaz podataka na Windows Form prikazu. U ovom programu je implementiran samo jedan prikaz unutar klase CompressionGraphics.

Na slici je moguće uočiti i folder **Resources** koji sadrži resurse koje program koristi. U ovom slučaju su to poruke koje je potrebno prikazivati korisniku.

Veoma važna klasa je i **DrawingUtility** koja je helper (pomoćna) klasa za iscrtavanje oblika i rad sa lokacijama na formi.

Zajednička logika

Osnovna funkcionalnost zajedničke logike koju koriste obje metode kompresije je metoda zadužena za dobijanje statistike o vjerovatnoći pojavljivanja karaktera u programu. Prikazana je sljedećim dijelom koda:

```
public static List<Character> GetProbabilityDictionary(string word)
{
    Counter = 0;
    List<Character> characters = new List<Character>();

    foreach (char c in word)
    {
        if (characters.Any(x => x.Char == c))
        {
            characters.Single(x => x.Char == c).Probability += 1f / word.Length;
        }
        else
        {
            characters.Add(new Character()
            {
                Char = c,
                Probability = 1f / word.Length
            });
        }
    }

    return characters.OrderByDescending(x => x.Probability).ToList();
}
```

Modeli

U ovom poglavlju će biti prikazani modeli koji su korišteni unutar programa. Cilj prikazivanja modela je uvid u način rada Logic klasa koje se predstavljaju u narednim poglavljima.

Najznačajniji model je model Node opisan u Node.cs i sljedećem kodu:

```
public class Node
{
        public char Char { get; set; }
        public Point Point { get; set; }
        public Size Size { get; set; }
        public Rectangle Rectangle { get; set; }
        public Node LeftChild { get; set; }
        public Node RightChild { get; set; }
        public string Code { get; set; }
        public bool IsRoot { get; set; } = false;
        public double Probability { get; set; }
        public int Index { get; set; }

        public Character MapToCharacter()
        {
            return new Character()
            {
                Char =  Char,
                Probability =  Probability
            };
        }

}
```

Pored osobina klase, objekti tipa Node sadržavaju i javnu metodu MapToCharacter koja služi za mapiranje (pretvaranje) objekta u tip Character. Taj model je opisan sljedećim kodom (on također sadrži metodu pretvorbe nazad u objekte tipa Node).

```
public class Character
{
        public char Char { get; set; }
        public double Probability { get; set; }


        public Node MapToNode()
        {
            return new Node()
            {
                Char = Char,
                Probability = Probability
            };
        }
}
```

Naredna dva fajla se sastoje od enumeracija potrebnih za rad aplikacije. To su CompressionMethod i Orientation koji su opisani narednim dijelovima koda:

```
public enum Orientation
{
    Root = 0,
    Left = 1,
    Right = 2
}
```

Orientation enumeracija zadužena je za opisivanje osobine čvora u odnosu na susjedne čvorove. Root je čvor koji se nalazi u ishodištu binarnog stabla, dok Left i Right predstavljaju orijentacije child čvorova unutar stabla.

```
public enum CompressionMethod
{
    Huffman = 1,
    ShanonFano = 2,
}
```

CompressionMethod enumeracija označava metodu kompresije koja je korištena. Ovaj podatak je izložen na korisnički interfejs, te ga je moguće izabrati iz menija.

Shanon-Fano Algoritam

Shanon-Fano kompresija podataka je opisana unutar ShanonFanoLogic klase. Klasa naslijeđuje BaseLogic klasu kako bi se koristile zajedničke osobine i iskoristila mogućnost za polimorfizam. Najvažnija metoda za kompresiju je metoda Compress. Implementacije su prikazane sljedećim dijelovima koda:

```
public static void Compress(ref List<Node> nodes, List<Character> characters)
{
    Node rootNode = new Node() { IsRoot = true };
    Compress(ref nodes, ref rootNode, characters, String.Empty, Orientation.Root);
}

private static void Compress(ref List<Node> nodes, ref Node parentNode, List<Character>
characters, string code, Orientation orientation)
{
    Node node = new Node()
    {
        Char = '\\',
        Code = code,
        LeftChild = null,
        RightChild = null,
        Index = Counter++
    };
    AssignNode(parentNode, node, orientation);
    nodes.Add(node);
    if (characters.Count == 1)
    {
        node.Char = characters[0].Char;
        return;
    }
    int splittingIndex = SplitArray(characters);
    Compress(ref nodes, ref node, characters.Take(splittingIndex).ToList(),
             $"{code}0", Orientation.Left);
    Compress(ref nodes, ref node, characters.Skip(splittingIndex).ToList(),
             $"{code}1", Orientation.Right);
}
```

Metode prikazuju realizaciju kompresiju Shanon-Fano algoritmom uz pomoćne metode AssignNode i SplitArray koje će biti prikazane u nastavku. Metoda koristi rekurziju kako bi nastavila kreaciju čvorova duž stabla, pri tom kreirajući novi list stabla svakom narednom iteracijom (što je prikazano početnim dijelom koda). Implementacija metode sa manje početnih parametara služi za inicijalni poziv i kreaciju ishodišta stabla. Nakon toga se poziva implementacija sa više parametara rekurzivno, dok se ne dođe do kraja binarnog stabla.

```
private static void AssignNode(Node parentNode, Node childNode, Orientation orientation)
{
    if (orientation == Orientation.Left)
    {
        parentNode.LeftChild = childNode;
    }
    else
    {
        parentNode.RightChild = childNode;
    }
}
```

AssignNode je pomoćna metoda koja dodjeljuje vrijednost enumeracije Orientation novim čvorovima dodanim u stablo.

```
private static int SplitArray(List<Character> characters)
{
    int index = 1;

    double upperArrayProbability = characters[0].Probability;
    double lowerArrayProbability = characters.Skip(index).Sum(x => x.Probability);
    double difference = Math.Abs(upperArrayProbability - lowerArrayProbability);

    for (; index < characters.Count; index++)
    {
        upperArrayProbability += characters[index].Probability;
        lowerArrayProbability -= characters[index].Probability;
        double newDifference = Math.Abs(upperArrayProbability - lowerArrayProbability);

        if (newDifference > difference)
        {
            return index;
        }
        else
        {
            difference = newDifference;
        }
    }
    return index;
}
```

SplitArray je metoda koja dijeli niz karaktera na dva niza na osnovu sume vjerovatnoća podnizova. Cilj je imati dva podniza karaktera sa što manjom razlikom između ukupnih vjerovatnoća, pri tome poštujući da se u nizu mogu naći samo karakteri koji su susjedni po vjerovatnoći (nakon što je urađeno sortiranje na osnovu vjerovatnoće u opadajućem poretku).

Huffman Algoritam

Huffmanov algoritam kompresije je realizovan unutar HuffmanLogic.cs fajla i HuffmanLogic klase koja nasljeđuje BaseLogic. Poput klase ShanonFanoLogic, i HuffmanLogic implementira kompresiju metodom Compress prikazanom u narednom dijelu koda:

```
public static void Compress(ref List<Node> nodes, List<Character> characters)
{
    List<Node> workingNodes = characters.Select(x => x.MapToNode()).ToList();

    while (workingNodes.Count != 1)
    {
        workingNodes = workingNodes.OrderBy(x => x.Probability).ToList();
        workingNodes.Add(MergeNodes(workingNodes[0], workingNodes[1]));
        workingNodes.RemoveRange(0,2);
    }

    AssignCodes(workingNodes[0], String.Empty);
    ExtractNodes(ref nodes, workingNodes[0]);
}
```

Metoda Compress iterativno prolazi kroz karaktere unutar niza sortiranog u opadajućem redoslijedu. Karakteri su inicijalno mapirani u čvorove I nalaze se u listi workingNodes. Nakon toga se u petlji broj čvorova u listi workingNodes smanjuje do jednog čvora koji će predstavljati ishodište cijelog binarnog stabla. To se postiže tako što se čvorovi postepeno dodjeljuju kao djeca drugih čvorova, a onda izbacuju iz liste. U tom trenutku se kreirao korijenski čvor za ta dva čvora i njega je potrebno dodati u listu. Ovo se postiže pomoćnom metodom MergeNodes.

```
private static Node MergeNodes(Node firstNode, Node secondNode)
{
    Node node = new Node()
    {
        Char = '\\',
        Probability = firstNode.Probability + secondNode.Probability,
        RightChild = firstNode,
        LeftChild = secondNode,
        Index = Counter++
    };
    return node;
}
```

MergeNodes pomoćna metoda kreira novi čvor koji predstavlja roditelja čvorovima koji su proslijeđeni metodi. Kao rezultat, taj čvor također ima i vjerovatnoću dobivenu sumiranjem vjerovatnoća proslijeđenih nodova. Privremeno mu je dodjeljen karakter '\' kako ne bi došlo do zabune da je neki drugi karakter iz izvornog seta podataka. Radi kasnijeg postepenog iscrtavanja, dodjeljena mu je i vrijednost brojača Counter, koji se također uvećava.

```
private static void AssignCodes(Node node, string code)
{
    node.Code = code;
    if (node.LeftChild != null)
    {
        AssignCodes(node.LeftChild, $"{code}0");
    }

    if (node.RightChild != null)
    {
        AssignCodes(node.RightChild, $"{code}1");
    }
}
```

Metoda AssignCodes rekurzivno navigira kroz stablo koristeći DFS algoritam i dodjeljuje kodove čvorovima iz stabla na osnovu kodova roditeljskih čvorova i njegove orijentacije.

```
private static void ExtractNodes(ref List<Node> nodes, Node node)
{
    nodes.Add(node);
    if (node.LeftChild != null)
    {
        ExtractNodes(ref nodes, node.LeftChild);
    }

    if (node.RightChild != null)
    {
        ExtractNodes(ref nodes, node.RightChild);
    }
}
```

Metoda ExtractNodes navigira kroz stablo rekurzivno koristeći DFS algoritam i dodaje sve čvorove binarnog stabla u listu nodes koja je proslijeđena kao referenca kako bi se mogla vratiti iz funkcije promjenjena.

DrawingUtility

Ova klasa je pomoćna klasa za iscrtavanje na Windows form prikazu. Metode nisu značajne za algoritme kompresije, ali će biti prikazane u svrhu dokumentovanja cijelog programa. U nastavu će biti prikazane sve metode i osobine klase DrawingUtility bez detaljnog objašnjenja njihove funkcionalnosti.

```
public static int Size { get; set; } = 30;
public static int DistanceX { get; set; } = 200;
public static int DistanceY { get; set; } = 70;
private static Font Font { get; set; }
private static Brush Brush { get; set; }
private static Brush CodeBrush { get; set; }
public static void InitializePositions(List<Node> nodes,
                                       int startingPositionX,
                                       int startingPoistionY)
{
    InitializePosition(nodes[0], startingPositionX, startingPoistionY, DistanceX);
}

public static void InitializePosition(Node node, int positionX, int positionY,
                                       int distance)
{
    node.Point = new Point(positionX, positionY);
    node.Size = new Size(Size,Size);
    distance = (int)(distance / 1.8f);
    if (node.LeftChild != null)
    {
        InitializePosition(node.LeftChild,
                           node.Point.X - distance,
                           node.Point.Y + DistanceY, distance);
    }

    if (node.RightChild != null)
    {
        InitializePosition(node.RightChild,
                           node.Point.X + distance,
                           node.Point.Y + DistanceY, distance);
    }
}
```

```
public static void DrawTreeNodes(List<Node> nodes, Graphics graphics, Pen pen, int i)
{
    Font = new Font("Arial", 15);
    Brush = new SolidBrush(Color.Black);
    CodeBrush = new SolidBrush(Color.Red);
    foreach (Node node in nodes)
    {
        if(node.Index < i)
        {
            DrawTreeNode(graphics, pen, node);
            ConnectNodes(graphics, pen, node, node.LeftChild);
            ConnectNodes(graphics, pen, node, node.RightChild);
        }
    }
}

public static void DrawTreeNode(Graphics graphics, Pen pen, Node node)
{
    node.Rectangle = new Rectangle(node.Point, node.Size);
    graphics.DrawEllipse(pen, node.Rectangle);
    graphics.DrawString(node.Char.ToString(),
                        Font,
                        Brush,
                        node.Point.X + Size/3.5f,
                        node.Point.Y);
}

private static void ConnectNodes(Graphics graphics,
                                 Pen pen,
                                 Node parentNode,
                                 Node childNode)
{
    if (parentNode != null && childNode != null)
    {
      Point startingPoint = new Point(parentNode.Point.X + Size / 2,
                                      parentNode.Point.Y + Size);

      Point endPoint = new Point(childNode.Point.X + Size / 2,
                                 childNode.Point.Y);

      Point midPoint = new Point((startingPoint.X + endPoint.X) / 2 - 5,
                                 (startingPoint.Y + endPoint.Y) / 2);

       graphics.DrawLine(pen, startingPoint, endPoint);
       graphics.DrawString(childNode.Code.Last().ToString(),
                           Font,
                           CodeBrush,
                           midPoint);
    }
}
```

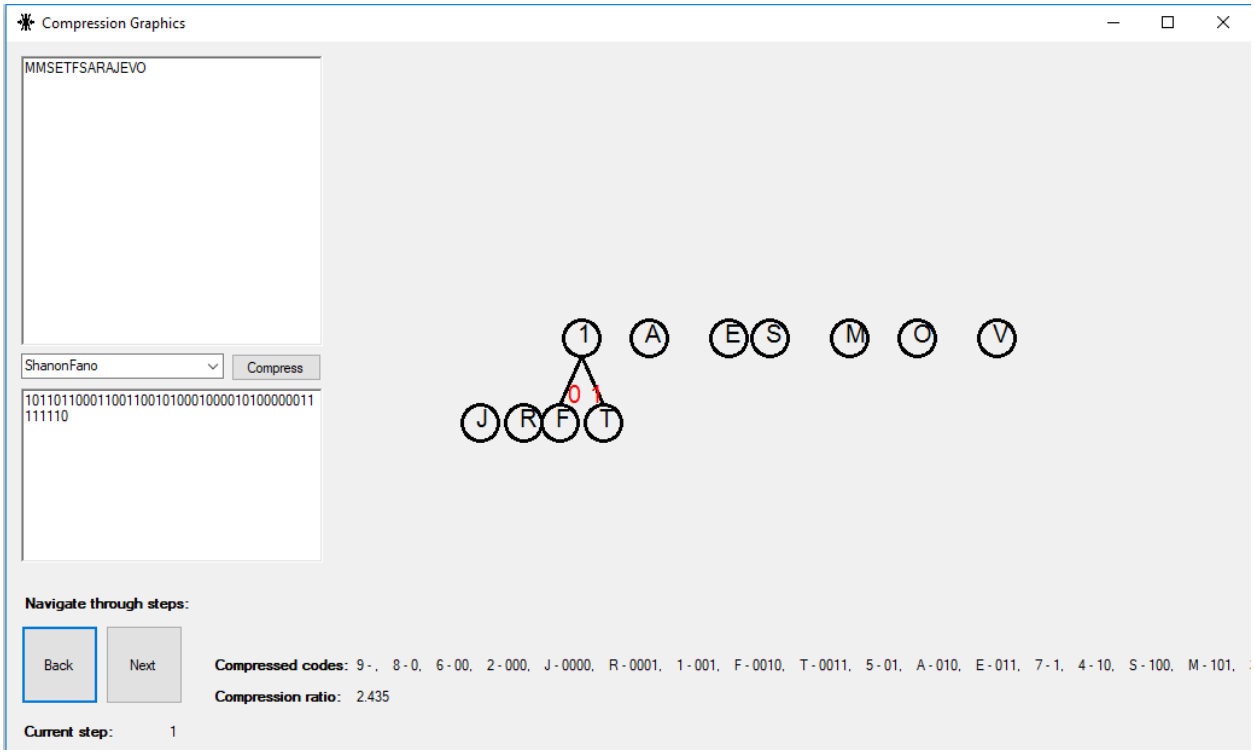
Grafički prikaz

Ovo poglavlje će prikazati program prilikom uobičajenog korištenja nekog korisnika. Na slikama je moguće uočiti prikaz binarnog stabla za algoritme ShanonFano i Huffman. Korisnički interfejs se sastoji od polja za unos podataka u gornjem lijevom uglu, glavnog polja za prikaz binarnog stabla, informacija o kodovima za kompresiju svakog karaktera (koji su također prikazani na binarnom stablu u crvenoj boji).

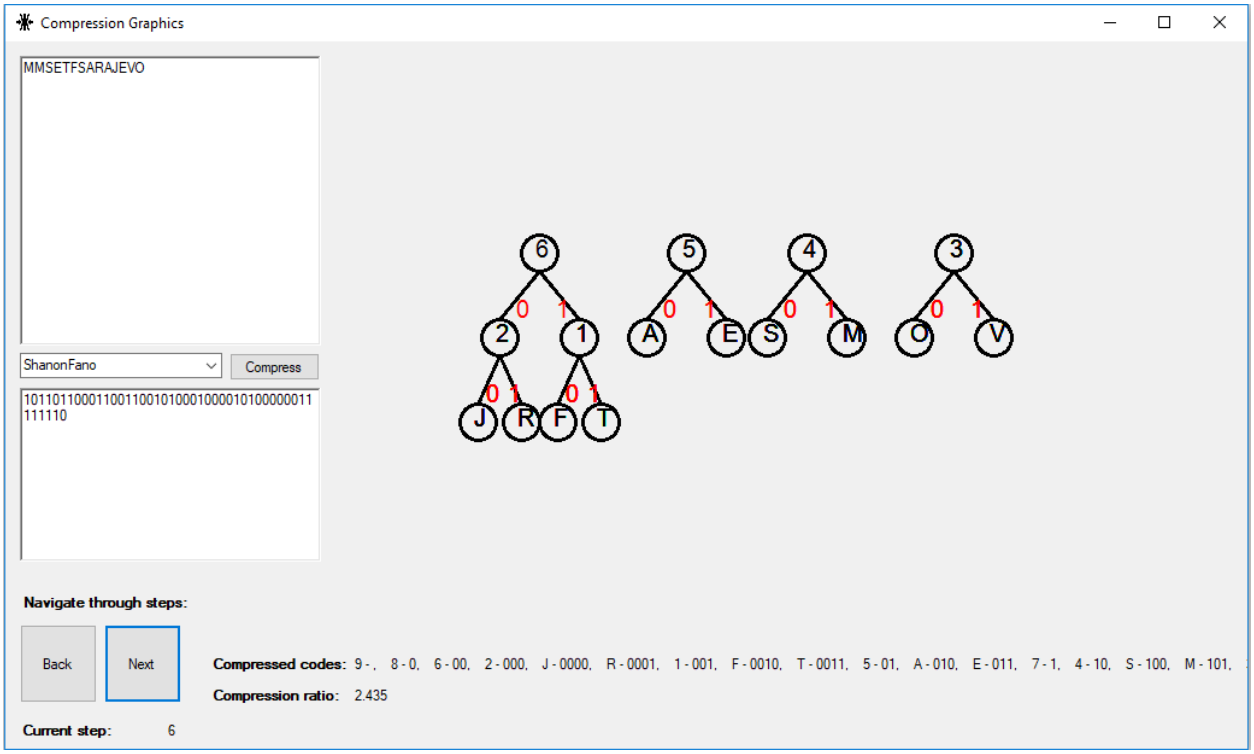
U donjem lijevom uglu je smještena navigacija za određivanje trenutnog koraka iscrtvanja stabla koji je prikazan na centralnom ekranu. Na lijevoj strani ekrana je također moguće izabrati i metodu kompresije iz padajućeg menija. Polje za prikaz teksta koje se nalazi ispod menija za izabir algoritma kompresije prikazuje komprimirani kod – tačnije kodove primjenjene za svaki od karaktera originalne poruke.

Shanon-Fano

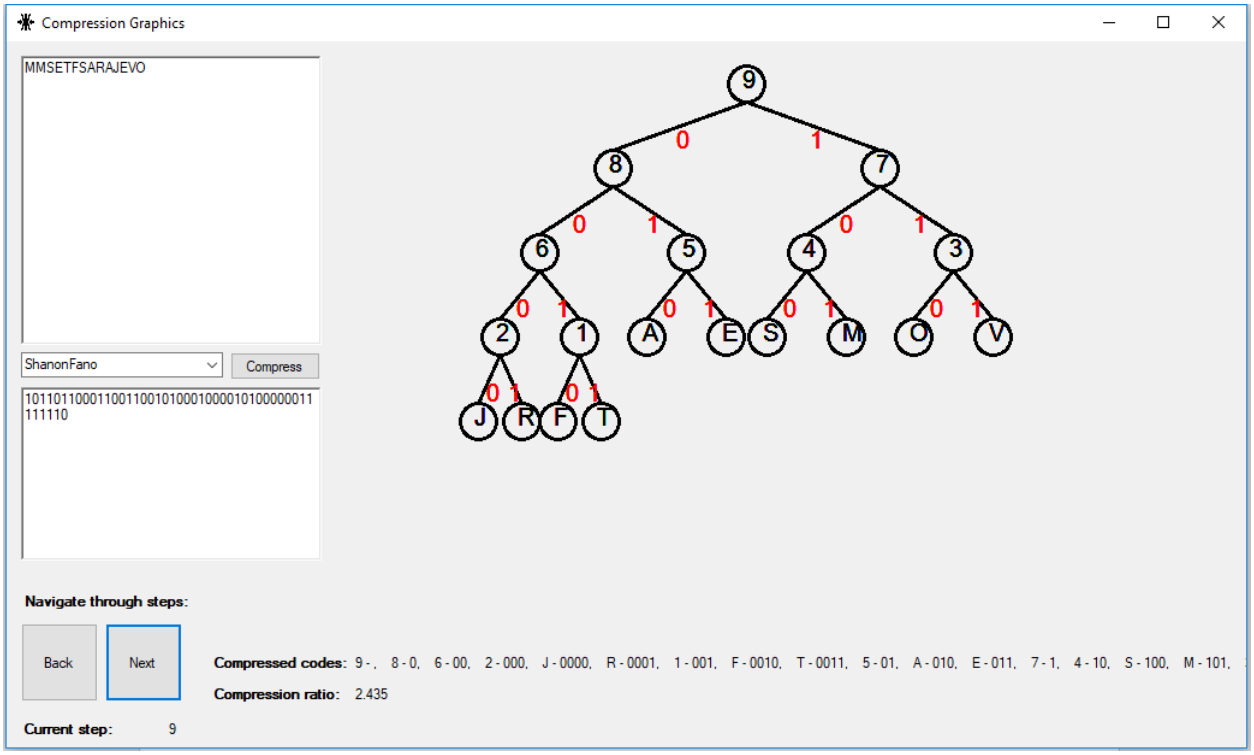
Na sljedećem nizu slika moguće je analizirati postupak kompresije Huffmanovim algoritmom i kreiranja binarnog stabla odlučivanja.



Slika 4 - ShanonFano kompresija, korak 1



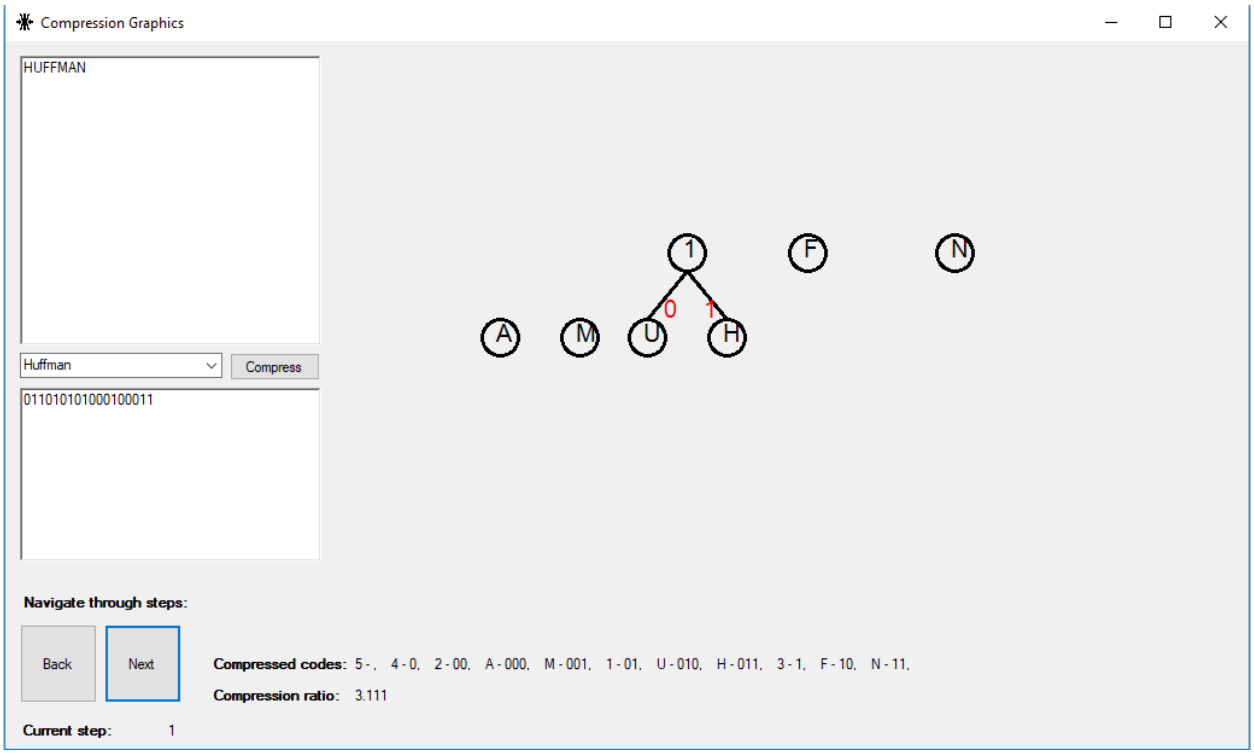
Slika 6 - ShanonFano kompresija, kroak 6



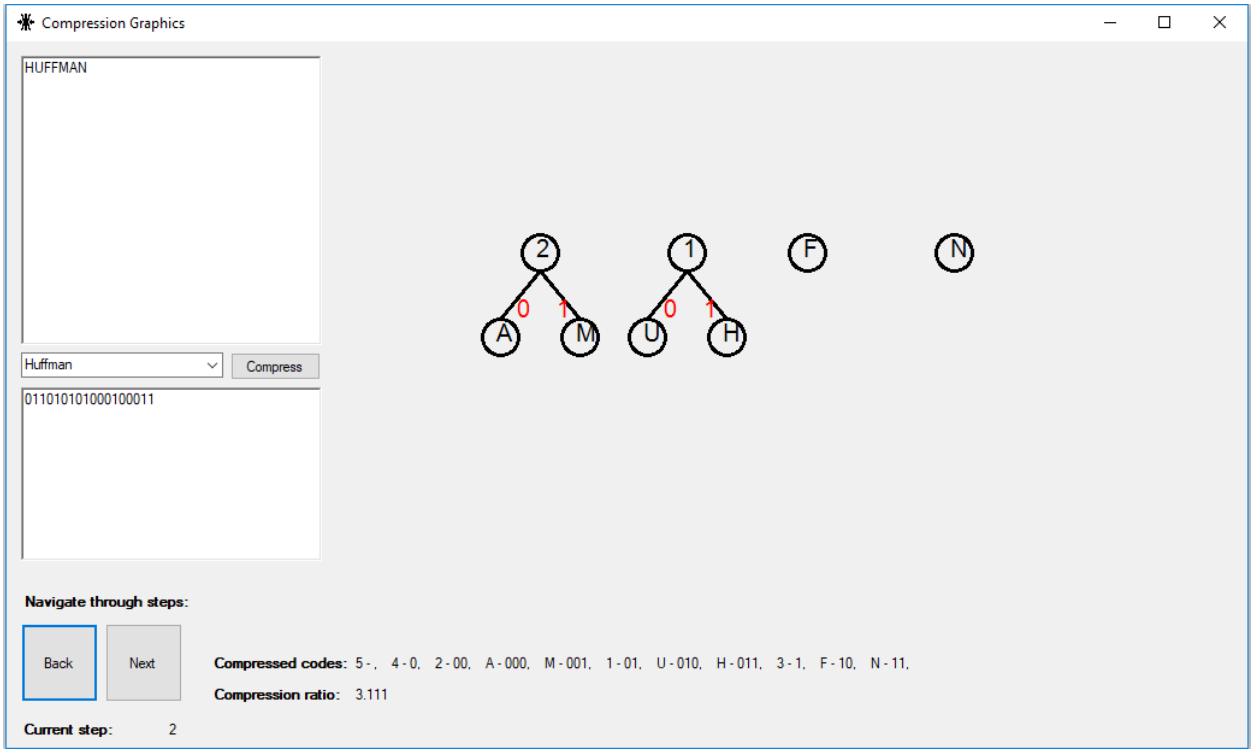
Slika 5 - ShanonFano kompresija, kroak 9

Huffman

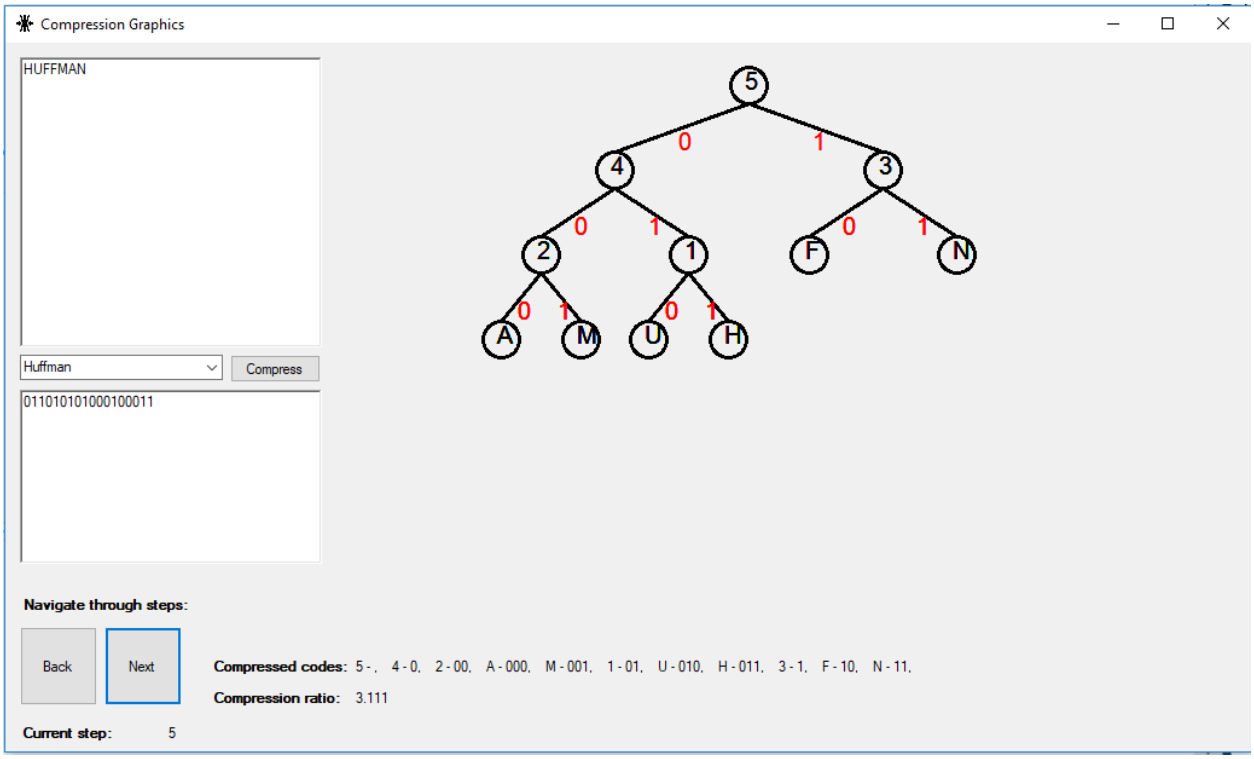
Na sljedećem nizu slika moguće je analizirati postupak kompresije Huffmanovim algoritmom i kreiranja binarnog stabla odlučivanja.



Slika 7 - Huffman kompresija, korak 1



Slika 9 - Huffman kompresija, korak 2



Slika 8 - Huffman kompresija, korak 5

Zaključak

U radu su predstavljene metode ShanonFano i Huffman za kompresiju podataka bez gubitaka. Dat je kratki teoretski uvod u ove metode, kao i općenito pojam kompresije podataka sa i bez gubitaka. Nakon toga su opisane metode praktično implementirane u kodu koji je prikazan u drugom poglavlju. Prikazan je i kod korišten za iscrtavanje oblika na ekranu i binarnog stabla kompresije. Konačno, prikazan je program na djelu, u slučajevima korištenja karakterističnim za običnog korisnika.

Rad ne treba da služi kao jedinstvena referenca i teoretska podloga za proučavanje kompresije podataka. Teoretske osnove koje su izložene u radu nisu dovoljne za duboko proučavanje već služe kao kratki uvod u ono što je potrebno implementirati u praktičnom dijelu. Zbog toga je količina detalja koji su prezentirani relativno mala.

Metode koje praktično implementiraju pseudokod predstavljen u teoretskoj osnovi nisu optimalne. Programski jezik koji je korišten (C#) također nije u potpunosti prilagođen čisto funkcionalnom programiranju koji bi možda ponudio bolje performanse algoritma kompresije. Ovo je objektno-orjentisani jezik, i zbog tih mogućnosti su korištene njegove osobine – algoritmi i program općenito se uveliko oslanjaju na rad sa objektima. Količina podataka koja se kreira prilikom izvršavanja algoritma, pripreme za iscrtavanje također nije optimalna. Kreira se mnogo podataka koji nisu neophodni za iscrtavanje, ali čine algoritam čitljivijim i lakšim za razumjevanje. Zbog toga algoritam koji je prezentovan ne bit rebalo koristiti za kompresiju realnih podataka (koji neće biti ograničeni u veličini) već se ograničiti na obrazovne benefite grafičkog prikaza metoda kompresije. Program može poslužiti prilikom provjere rezultata primjene algoritma na papiru ili u edukativne svrhe za nekoga ko još uvijek nije ovladao algoritmima kompresije.

Buduća poboljšanja programa u svakom slučaju mogu da budu fokusirana na optimizaciju rada sa podacima, brzine izvršavanja, kao i izbacivanja metoda i modela koji nisu neophodni za rad. Pored toga, moguće je napraviti velika poboljšanja u izgledu binarnog stabla, čitljivosti prikazanih podataka, kao i detaljnosti koraka koji se ispisuju postepeno. Jedno od mogućih poboljšanja je i ispisivanje niza karaktera sa kojim algoritmi rade u svakom koraku svoga izvršavanja.

Literatura

[1] Predavanja za predmet Multimedijalni Sistemi, akademska 2018/2019, H. Šupić

[2] Predavanja za predmet Diskretna Matematika , akademska 2018/2019 Ž. Jurić

[3] C# 6.0 in a Nutshell: The Definitive Reference, Joseph Albahari, Ben Albahari, O'Reilly Media, Inc

[4] data compression (https://searchstorage.techtarget.com/definition/compression)

[5] How does File Compression Work (https://www.makeuseof.com/tag/how-does-file-compression-work/)

[6]File Compression (http://computing.homedns.org/Tests/National6_ISDD/CourseContent/MediaTypes/Compression.php)

[7] Static defined word schemes (https://www.ics.uci.edu/~dan/pubs/DC-Sec3.html)

[8] A memory-efficient and fast Huffman decoding algorithm, Hong-Chung Chen, Yue-Li Wang, Yu-Feng Lan

[9] Study on Data Compression Technique, Md. Jayedul Haque, Mohammad Nurul Huda

[10] Data compression using Shannon-fano algorithm, Mr. Mahesh Vaidya, Mr.Ekjot Singh Walia, Mr. Adtya Gupta

Popis slika