

Elektronski fakultet

Univerzitet u Nišu



Metodi i sistemi za obradu signala

**Kompresija podataka primenom adaptivnog Huffman-ovog
kodiranja: A – varijanta**

Profesor:

Prof. dr Miloš Radmanović

Student:

Miloš Stanojević 18399

Sadržaj

Kompresija podataka	3
Kompresija bez gubitaka	4
Kompresija sa gubicima	6
Huffmanovo-ovo kodiranje	6
Huffman-ovo dekodiranje	7
Adaptivno Huffman-ovo kodiranje	8
Adaptivno Huffman-ovo kodiranje – A varijanta	9
Objašnjenje implementacije kompresije A varijante adaptivnog Huffman-ovog kodiranja	10
- Klasa HuffmanCode	10
- Main program	15
Primeri pokretanja implementacije	17
Zaključak	19
Reference	19

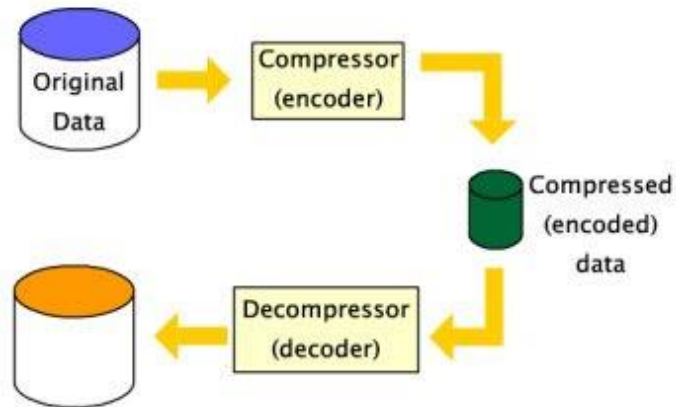
Kompresija podataka

Kompresija podataka je proces smanjenja veličine digitalnih podataka kako bi se optimizovala potrebna memorija za njihovo skladištenje kao i optimizovao prenos tih podataka kroz mrežu. Osnovni cilj kompresije podataka je smanjenje veličine podataka bez gubitka bitnih informacija, čime se postiže ušteda u prostoru ili povećanje brzine prenosa podataka. Ovaj proces se koristi u različitim aplikacijama, kao što su digitalna arhiviranja, prenos medija putem interneta, komunikacija mobilnih uređaja i mnoge druge oblasti.

Postoje dva osnovna tipa kompresije podataka: kompresija podataka sa gubicima i kompresija podataka bez gubitaka. Kompresija podataka sa gubicima je vrsta kompresije kod koje su prihvatljivi određeni gubici u informacijama. Kod ove kompresije nije garantovano da će podaci nakon kompresije biti isti kao oni sa početka. Ovim putem bi se dobio podatak koji je skoro identičan, ali bi ipak neke informacije koje “nisu bitne” nedostajale. Ovaj vid kompresije se najčešće koristi prilikom rada sa slikama i video fajlovima. Kompresija podataka bez gubitaka omogućava potpun oporavak originalnih podataka bez gubitaka, ali može imati manju stopu kompresije u poređenju sa kompresijom sa gubicima.

Algoritmi za kompresiju podataka se mogu klasifikovati prema različitim kriterijumima, kao što su metode kompresije, tipovi podataka koji se obrađuju ili načinima pomoću kojih postižu kompresiju. Neke od najpoznatijih tehnika kompresije uključuju Huffman-ovo kodiranje, LZ algoritme (poput LZ77 i LZ88), aritmetičko kodiranje, kao i transformacione metode poput Discrete Cosine Transform (DCT) korišćene u JPEG kompresiji ili Discrete Wavelet Transform (DWT) koja se koristi u JPEG2000 ili kompresiji video sadržaja.

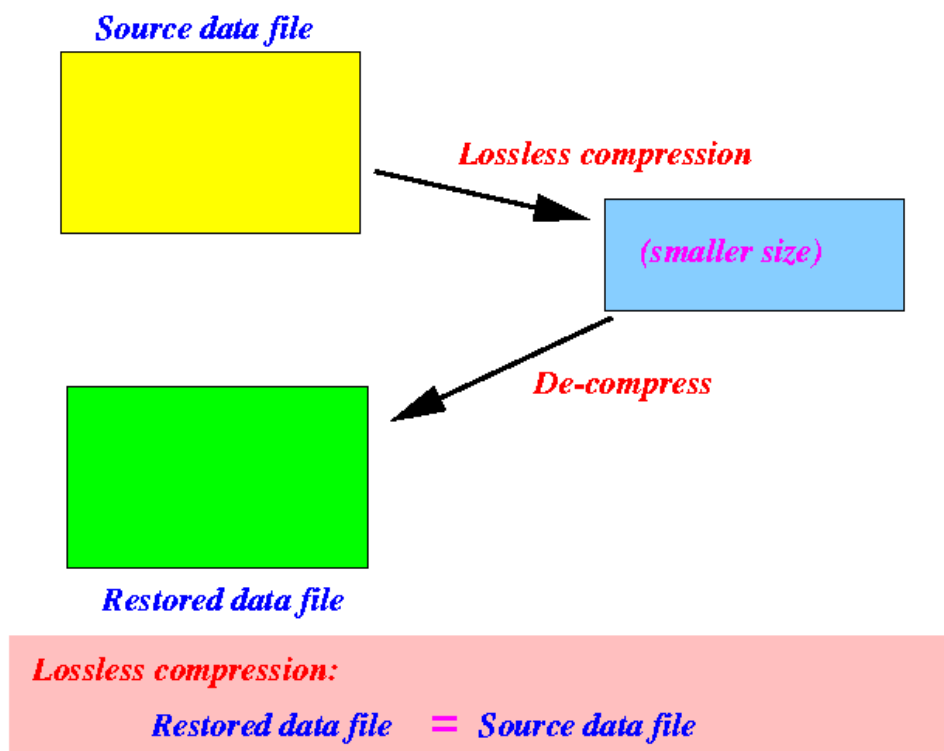
Kompresija podataka igra ključnu ulogu u modernim tehnologijama, omogućavajući efikasno upravljanje ogromnim količinama digitalnih podataka. Razumevanje principa i tehnika kompresije podataka ključno je za optimizaciju performansi i efikasnost mnogih digitalnih sistema i aplikacija.



Kompresija bez gubitaka

Algoritmi kompresije bez gubitaka obično iskorišćavaju statičku redundanciju za prikaz podataka bez gubitka informacija, što proces čini reverzibilnim. Ovo je ključna osobina za aplikacije gde je očuvanje svih detalja originalnih podataka od vitalnog značaja, kao što su medicinske slike, tekstualni dokumenti, ili softverski kodovi. Osnovni princip kompresije bez gubitaka je identifikacija i eliminacija redundantnih informacija ili statičkih uzoraka koji se mogu efikasno predstaviti manjim brojem podataka.

Jedna od popularnijih tehnika kompresije bez gubitaka je Huffman-ovo kodiranje, koje se često koristi za kompresiju teksta. Ovaj algoritam dodeljuje kraće kodove čestim simbolima i duže kodove retkim simbolima, čime se postiže efikasna kompresija bez gubitka informacija. Pored Huffman-ovog kodiranja, tu su i različite metode složenijih algoritama poput LZ77, LZ78 i aritmetičkog kodiranja.



Na sledećoj slici možemo video sliku pre i nakon primene kompresije bez gubitaka.



Kompresija sa gubicima

Kompresija sa gubicima ili nepovratna kompresija je vrsta kompresije podataka koja koristi neprecizne aproksimacije i odbacivanje dela podataka da bi predstavila sadržaj. Ove tehnike se koriste za smanjenje veličine podataka za skladištenje, manipulaciju i prenos sadržaja. Veći stepeni aproksimacije stvaraju veće gubitke informacija (grublje podatke). Količina smanjenja podataka moguća primenom kompresije sa gubicima mnogo je veća nego primenom kompresije bez gubitaka.

Dobro dizajnirana tehnologija kompresije sa gubicima često značajno smanjuje veličine fajlova pre nego što degradacija bude primećena od strane krajnjeg korisnika. Čak i kada je primetna od strane korisnika, dalje smanjenje podataka može biti poželjno (na primer u real-time komunikacijama). Najkorišćeniji algoritam koji primenjuje kompresiju sa gubicima je Discrete Cosine Transform (DCT) algoritam. Ova vrsta kompresije se najčešće koristi za kompresiju zvuka, snimaka i slika, pogotovo u aplikacijama za razgovore preko interneta.

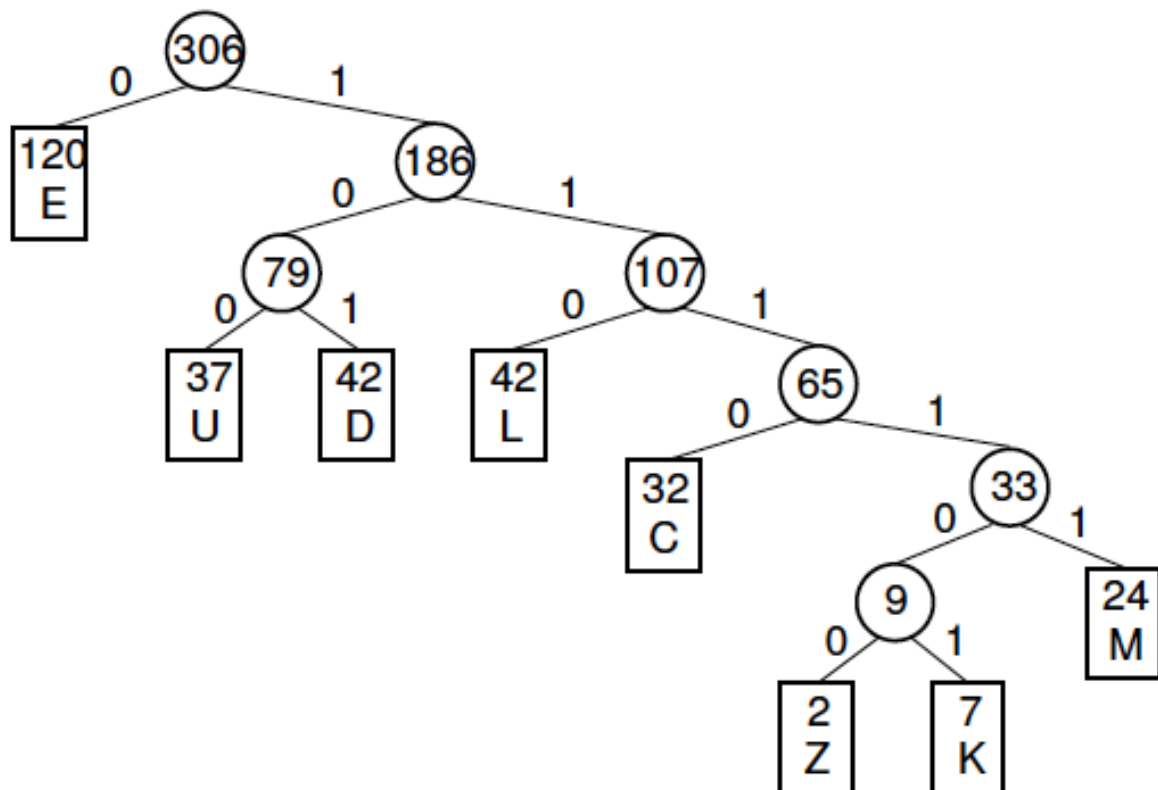


Na slici je prikazan primer kompresije sa gubicima, zavisno od stepena kompresije.

Huffman-ovo kodiranje

Huffman-ovo kodiranje je jedna od osnovnih tehnika kompresije podataka, koja se često koristi za efikasno kodiranje simbola sa različitim učestalostima pojavljivanja. Ovaj algoritam, koji je razvio David A. Huffman 1952. godine, zasniva se na konceptu izgradnje optimalnog binarnog prefiksnog stabla, gde se čestim simbolima dodeljuju kraći kodovi, dok se ređim simbolima dodeljuju duži kodovi.

Metod počinje tako što se na početku gradi lista svih simbola azbuke poređanih u opadajućem redosledu po njihovoj verovatnoći pojavljivanja. Nakon toga, gradi se stablo, od dole na gore, kod koga svaki list predstavlja jedan simbol. Ovaj postupak se odvija u koracima, gde u svakom koraku dva simbola sa najmanjom verovatnoćom budu odabrani, dodani na vrh delimičnog stabla, obrisani iz liste i zamenjeni pomoćnim simbolima koji predstavljaju svaki od njih. Kada u listi ostane samo jedan pomoćni simbol (koji predstavlja čitavu azbuku), stablo je kompletno. Prolaskom kroz stablo dobija se kod za svaki od simbola.



Huffman-ovo dekodiranje

Huffman-ovo dekodiranje je obrnuti proces od Huffman-ovog kodiranja i omogućava rekonstrukciju originalnih podataka iz kodiranih binarnih nizova. Ključni korak u Huffman-ovom dekodiranju je rekonstrukcija binarnog prefiksnog stabla koje se koristilo prilikom kodiranja. Ova struktura je presudna za efikasno mapiranje kodova nazad u originalne simbole.

Da bi se dekodirali kodovi, počinje se od korena binarnog stabla i prati se put kroz stablo na osnovu niza bitova iz kodiranog niza. Svaki put kroz stablo vodi do jednog od listova, koji predstavlja jedan od originalnih simbola. Na taj način, prolaskom kroz stablo za svaki kod, možemo efikasno dekodirati ceo niz simbola.

Efikasnost dekodiranja leži u tome što je stablo binarno i optimalno, što znači da je svaki kod jedinstven i neizborno mapiran na određeni simbol. Ovaj proces omogućava brzo i pouzdano dekodiranje podataka kodiranih Huffman-ovim algoritmom, čime se omogućava potpuna rekonstrukcija originalnih podataka bez gubitaka.

Adaptivno Huffman-ovo kodiranje

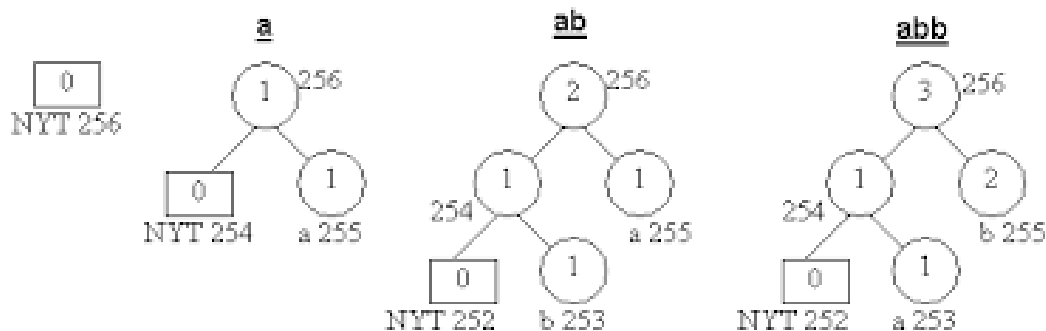
Adaptivno Huffman-ovo kodiranje, poznato još i kao Dinamičko Huffman-ovo kodiranje, omogućava dinamičko prilagođavanje kodiranja tokom procesa kompresije. Ovaj algoritam se koristi u situacijama gde učestalost pojavljivanja simbola može varirati tokom vremena ili gde je skup podataka dinamičan i nepredvidiv.

Glavna ideja je da kompresor i dekompresor počnu sa praznim stablom i da ga menjaju tokom čitanja simbola iz podataka. Kompresor i dekompresor bi trebalo da menjaju stablo na isti način, tako da u bilo kom trenutku koriste iste kodove, iako ti kodovi mogu da se menjaju od koraka do koraka.

Kompresor na početku počinje sa praznim stablom i ni jednom simbolu nije dodeljen kod. Prvi simbol iz unosa je jednostavno ispisan na izlazu bez da je kompresovan. Nakon toga je taj simbol dodat u stablo i dodeljen mu je kod. Sledeći put kada se taj simbol nađe na ulazu, njegov trenutni kod će biti ispisan na izlazu i njegova frekvencija pojavljivanja će biti povećana za jedan. S' obzirom da ovo menja stablo, potrebno je proveriti da li je stablo još uvek Huffman-ovo stablo, ako nije, potrebno ga je srediti, što dovodi do promene kodova.

Dekompresor izvršava iste korake, samo u obrnutom redosledu. Kad čita nekompresovani simbol, dodaje ga u stablo i dodeljuje mu kod. Kad čita kod u njegovoj kompresovanoj formi (promenljive dužine), koristi trenutno stablo da otkrije kom simbolu odgovara taj kod i modifikuje stablo na isti način na koji to radi i kompresija.

Adaptivno kodiranje je ključno za mnoge aplikacije koje zahtevaju dinamičko i efikasno upravljanje podacima, kao što su prenos podataka u realnom vremenu i slično.



Adaptivno Huffman-ovo kodiranje – A varijanta

Ova metoda adaptivnog Huffman-ovog kodiranja je jednostavnija, ali manje efikasna. Glavna ideja je da se izračuna set od n kodova promenljive veličine koji su zasnovani na jednakim šansama pojavljivanja, a da nakon toga n simbola dobije te kodove u nasumičnom redosledu, pa da se dodeljeni kodovi za simbole menjaju u toku čitanja simbola sa ulaza. Za razliku od klasičnog adaptivnog Huffman-ovog kodiranja, ovaj metod ne koristi stablo, već tabelu, koja se sastoji od tri kolone, što ga čini bržim od klasičnog metoda, jer je, umesto manipulacija stablom, potrebna jednostavna manipulacija tabelom.

Prva kolona predstavlja promenljivu kojom je označen simbol. Druga kolona predstavlja broj pojavljivanja (frekvencu pojavljivanja) datog simbola u dosad pročitanoj ulazu. Treća kolona predstavlja kod kojim se kodira dani simbol.

Prilikom čitanja ulaza ovu tabelu je potrebno sortirati u opadajućem redosledu po frekvenci pojavljivanja simbola. Međutim potrebno je u tabeli sortirati samo prve dve kolone, dok treća ostaje nepromenjena, čime se osigurava da će simbol sa najvećom frekvencijom pojavljivanja biti kodiran najkraćim kodom.

Name	Count	Code	Name	Count	Code	Name	Count	Code	Name	Count	Code
a_1	0	0	a_2	1	0	a_2	1	0	a_4	2	0
a_2	0	10	a_1	0	10	a_4	1	10	a_2	1	10
a_3	0	110	a_3	0	110	a_3	0	110	a_3	0	110
a_4	0	111	a_4	0	111	a_1	0	111	a_1	0	111
(a)			(b)			(c)			(d)		

Slika predstavlja primer primene ove varijante za ulaz "a2 a4 a4".

Objašnjenje implementacije kompresije A varijante adaptivnog Huffman-ovog kodiranja

Algoritam je implementiran pomoću jezika C#. Najbitnije funkcionalnosti ove implementacije biće objašnjene u nastavku.

Klasa **HuffmanCode**

- U ovoj klasi implementirane su sve funkcije potrebne za uspešnu implementaciju kompresije i dekompresije pomoću adaptivnog Huffman-ovog kodiranja.
- Na slici možemo videti deklaraciju struktura podataka koje ćemo u nastavku koristiti.

```
public Dictionary<char, Tuple<int, string>> _codeTable;  
public Dictionary<char, int> frequencyMap = new Dictionary<char, int>();  
public List<string> codes = new List<string>();  
public string encodedTextWhole = "";
```

- Promenljiva **_codeTable** je struktura koja služi za predstavljanje kodne table koja se u ovom algoritmu sastoji od tri kolone u kojoj prva predstavlja simbol koji se kodira, druga frekvencu pojavljivanja tog simbola, a treća kod za taj simbol.
- Promenljiva **frequencyMap** je pomoćna struktura koja služi kao pomoć pri pamćenju frekvence pojavljivanja svakog karaktera.
- Promenljiva **codes** je pomoćna lista u kojoj se čuvaju kodovi za svaki simbol u redosledu u kom se nalaze u inicijalnoj kodnoj tabeli.
- Na sledećoj slici prikazana je implementacija funkcije **InitializeCodeTable()**. Funkcija na početku kreira listu sa ASCII simbolima, ali tako da je redosled simbola uvek nasumičan. Nakon toga (u **for** petlji) za svaki od karaktera generiše kod pozivom funkcije **GenerateCode**, ubacuje karakter u kodnu tabelu (**_codeTable**) zajedno sa njegovim kodom, postavljajući frekvencu pojavljivanja tog karaktera na 0. Nakon toga inicijalizuje pomoćne strukture **frequencyMap** i **codes**.

```
//Funkcija za postavljanje inicijalne tabele sa kodovima za sve karaktere
1 reference
public void InitializeCodeTable()
{
    _codeTable = new Dictionary<char, Tuple<int, string>>();

    List<char> asciiChars = Enumerable.Range(32, 127 - 32 + 1).Select(i => (char)i).ToList();
    asciiChars.Add('\r');
    asciiChars.Add('\n');

    Random rng = new Random();
    int n = asciiChars.Count;
    while (n > 1)
    {
        n--;
        int k = rng.Next(n + 1);
        char value = asciiChars[k];
        asciiChars[k] = asciiChars[n];
        asciiChars[n] = value;
    }

    for(int i=0;i<asciiChars.Count;i++)
    {
        string code = GenerateCode(i+1);
        _codeTable[asciiChars[i]] = new Tuple<int, string>(0, code);
        frequencyMap[asciiChars[i]] = 0;
        codes.Add(code);
    }
}
```

- Na slici je prikazana funkcija **GenerateHuffmanCodes**, koja za svaki prosleđeni karakter iz teksta (ili sa ulaza) uređuje kodnu tabelu potencijalno menjajući kod simbola u njoj. Najpre povećava frekvencu pojavljivanja primljenog karaktera u pomoćnoj strukturi **frequencyMap**, nakon čega se poziva funkcija **SortTable**, koja sortira čitavu kodnu tabelu.

```
//Funkcija za odredjivanje koda za svaki karakter u tekstu
1 reference
public void GenerateHuffmanCodes(char c)
{
    frequencyMap[c]++;

    SortTable();

    string encodedText = _codeTable[c].Item2;
    string invertedEncodedText = new string(encodedText.Reverse().ToArray());
    encodedTextWhole += invertedEncodedText;
    Console.WriteLine($"Symbol: {c}, Current Code: {encodedText}");

    string decodedSymbol = DecompressBinaryInput(encodedText);
    Console.WriteLine($"Decompressed Symbol: {decodedSymbol}");
}
```

- Na sledećoj slici prikazana je implementacija funkcije **DecompressBinaryInput** koja više služi kako bi se prikazao rad samog algoritma, ili za prikaz slučaja kada bi se prikazivala korist ovog algoritma u prenosu podataka. Ona prihvata binarnu

kodiranu vrednost trenutno poslatog karaktera i pokušava da nađe poklapanje sa kodnom tabelom, nakon čega prikazuje dekodiranu vrednost u konzoli.

```
1 reference
public string DecompressBinaryInput(string binaryInput)
{
    StringBuilder decodedText = new StringBuilder();
    StringBuilder currentCode = new StringBuilder();
    foreach (char bit in binaryInput)
    {
        currentCode.Append(bit);
        foreach (var kvp in _codeTable)
        {
            if (kvp.Value.Item2 == currentCode.ToString())
            {
                decodedText.Append(kvp.Key);
                currentCode.Clear();
                break;
            }
        }
    }
    Console.WriteLine();
    Console.WriteLine("Decoded text: " + decodedText);
    Console.WriteLine();
    return decodedText.ToString();
}
```

- Sledeća funkcija ima vrlo jednostavnu implementaciju, ona jednostavno kreira kod za svaki karakter po pravilu za kodiranje pomoću ovog algoritma (karakter sa najvećom frekvencom pojavljivanja ima kodnu vrednost 0, a svaki sledeći karakter redom dobija po jednu 1 za onoliko mesta koliko je udaljen od najčešćeg karaktera).

```

//Funkcija za kreiranje koda za svaki simbol
1 reference
private string GenerateCode(int length)
{
    string code = "";
    if (length == 1)
    {
        code = "0";
        return code;
    }
    for (int i = 0; i < length - 1; i++)
    {
        code += "1";
    }
    code += "0";
    return code;
}

```

- Funkcija na sledećoj slici implementira funkcionalnost kojom se od potpuno kodirane poruke dobija originalna poruka (dekompresija). Dekompresiju je potrebno izvršiti tako što se primljena (pročitana) sekvenca čita unazad. Uzima se bit po bit kodirane sekvence i pokušava da se nađe kod koji odgovara toj sekvenci. U slučaju da se za trenutno izdvojenu podsekvencu u tabeli ne nalazi kod koji za karakter koji joj odgovara, podsekvenci se dodaje i sledeći bit iz kodiranog ulaza. Kada se nađe poklapanje vrši se smanjivanje frekvence pojavljivanja učitanoog karaktera u pomoćnoj strukturi **frequencyMap**, pomoću koje se i čitava kodna tabela ponovo sortira pozivom funkcije **SortTable**.

```

//Funkcija za dekodiranje kodiranih podataka
1 reference
public string DecompressText(string encodedText)
{
    string decompressedText = "";
    int endIndex = encodedText.Length-1;

    while (endIndex >= 0)
    {
        string currentCode = "";
        bool codeFound = false;

        //Citanje koda bit po bit od pozadi
        for (int i = endIndex; i >= 0; i--)
        {
            currentCode += encodedText[i];

            //Provera da li trenutni kod (currentCode) postoji u kodnoj tabeli
            foreach (var kvp in _codeTable)
            {
                if (kvp.Value.Item2 == currentCode)
                {
                    decompressedText = kvp.Key + decompressedText;
                    frequencyMap[kvp.Key]--;
                    endIndex = i - 1;
                    codeFound = true;

                    SortTable();

                    break;
                }
            }

            //Prekini petlju ako je pronadjen kod
            if (codeFound)
                break;

            //Predji na sledeci bit ako nije pronadjen kod
            if (!codeFound)
                endIndex--;
        }

        return decompressedText;
    }
}

```

- Na sledećoj slici prikazana je implementacija funkcije za sortiranje kodne tabele **SortTable**. Pomoću prerađenog delegata **Sort** vrši se sortiranje liste **sortedList**, koja sadrži čitavu kodnu tabelu, tako što se kreira lambda izraz koji za parametre ima dva key-value para čije frekvence pojavljivanja najpre upoređuje i rezultat smešta u promenljivu **freqComparison**. Izraz u **return** delu razrešava slučaj kada su frekvence pojavljivanja karaktera jednake tako što upoređuje vrednosti samih karaktera u ASCII tabeli i vraća to kao rezultat. Na dalje se jednostavno kreira trenutna tabela za čuvanje sortirane kodne table, koja se u ovom trenutku nalazi u listi **sortedList**, zbog ograničenja prilikom korišćenja **Tuple** tipa podatka koji zabranjuje menjanje vrednosti.

```

2 references
private void SortTable()
{
    var sortedList = _codeTable.ToList();

    sortedList.Sort((x, y) =>
    {
        int freqComparison = frequencyMap[y.Key].CompareTo(frequencyMap[x.Key]);
        return freqComparison == 0 ? x.Key.CompareTo(y.Key) : freqComparison;
    });

    Dictionary<char, Tuple<int, string>> newCodeTable = new Dictionary<char, Tuple<int, string>>();
    for (int j = 0; j < sortedList.Count; j++)
    {
        char symbol = sortedList[j].Key;
        string code = _codeTable[symbol].Item2;
        newCodeTable[symbol] = new Tuple<int, string>(frequencyMap[symbol], codes[j]);
    }

    _codeTable = newCodeTable;
}

```

Main program

- U main programu se kreira objekat klase **HuffmanCode**, pomoću koga se pozivaje sve prethodno opisane funkcije neophodne za demonstraciju rada adaptivnog Huffman-ovog algoritma u A varijanti. Najpre se poziva funkcija za inicijalizaciju tabele koja se čuva u svom tekstualnom fajlu. Nakon toga se korisniku daje izbor da li će sam dati input za kompresiju ili će se kompresija izvršiti nad sadržajem tekstualnog fajla "input.txt". Nakon toga se za svaki od pročitanih (unetih) simbola najpre vrši kodiranje tog karaktera funkcijom **GenerateHuffmanCodes**, a nakon toga i čuvanje nove (sortirane) kodne tabele u fajl "code_table.txt". Nakon uspešno izvršene kompresije (kodiranja) rezultat se prikazuje u konzoli, a kodirana sekvenca se čuva u binarnom fajlu "encoded_text.bin".

```

class Program
{
    0 references
    static void Main(string[] args)
    {
        HuffmanCode huffman = new HuffmanCode();
        string initialCodeTable = "initial_code_table.txt";
        string codeTableFilePath = "code_table.txt";
        string binaryFile = "encoded_text.bin";

        huffman.InitializeCodeTable();
        huffman.WriteHuffmanCodesToFile(initialCodeTable);

        Console.WriteLine("Should the input be read from the file? (Y/N)");
        string choice = Console.ReadLine();

        string input = "";

        if (choice.ToUpper() == "Y")
        {
            input = huffman.ReadFile("input.txt");
        }
        if (choice.ToUpper() == "N")
        {
            Console.WriteLine("Enter the text to compress and decompress (type 'quit' to exit):");
            input = Console.ReadLine();
        }

        string encodedText = "";

        foreach (char ch in input)
        {
            huffman.GenerateHuffmanCodes(ch);
            huffman.WriteHuffmanCodesToFile(codeTableFilePath);
            Console.WriteLine();
            encodedText += ch;
        }

        Console.WriteLine("Decompressed inverse text: " + huffman.DecompressText(huffman.GetWholeEncodedText()));

        huffman.ConvertToBytesAndWriteToFile(huffman.GetWholeEncodedText(), binaryFile);

        Console.WriteLine("End text: " + encodedText);
        Console.WriteLine("Whole encoded text: " + huffman.GetWholeEncodedText());
    }
}










```


Primeri pokretanja implementacije

Na sledećim slikama možemo videti primer pokretanja programa u slučaju kada je korisnik taj koji zadaje ulaz. U konzoli se prikazuje trenutni kod koji karakter ima u kodnoj tabeli, a ujedno i prikaz isto tog simbola nakon dekodiranja (za slučaj kada se ovaj algoritam koristi za prenos poruka). Na kraju vidimo uspešno dekodiran tekst, tekst koji je korisnik uneo, kao i kodnu sekvencu kojom je kodiran ulaz.

```
Should the input be read from the file? (Y/N)
N
Enter the text to compress and decompress (type 'quit' to exit):
Kratak primer.
Symbol: K, Current Code: 0
Decoded text: K
Decompressed Symbol: K
Symbol: r, Current Code: 10
Decoded text: r
Decompressed Symbol: r
Symbol: a, Current Code: 10
Decoded text: a
Decompressed Symbol: a
Symbol: t, Current Code: 1110
Decoded text: t
Decompressed Symbol: t
Symbol: a, Current Code: 0
Decoded text: a
Decompressed Symbol: a
Symbol: k, Current Code: 110
Decoded text: k
Decompressed Symbol: k
Symbol: , Current Code: 10
Decoded text:
Decompressed Symbol:
Symbol: p, Current Code: 11110
Decoded text: p
Decompressed Symbol: p
Symbol: r, Current Code: 10
Symbol: p, Current Code: 11110
Decoded text: p
Decompressed Symbol: p
Symbol: r, Current Code: 10
Decoded text: r
Decompressed Symbol: r
Symbol: i, Current Code: 11110
Decoded text: i
Decompressed Symbol: i
Symbol: m, Current Code: 1111110
Decoded text: m
Decompressed Symbol: m
Symbol: e, Current Code: 11110
Decoded text: e
Decompressed Symbol: e
Symbol: r, Current Code: 0
Decoded text: r
Decompressed Symbol: r
Symbol: ., Current Code: 1110
Decoded text: .
Decompressed Symbol: .
Decompressed text: Kratak primer.
End text: Kratak primer.
Whole encoded text: 0010101110011010111101011110111110111100111
```

Sledeća slika prikazuje fajlove koji su kreirani nakon pokretanja programa za slučaj kada se kompresuje sadržaj fajla umesto korisničkog unosa. Može se primetiti da je razlika u veličini početnog fajla ("input.txt") i kompresovanog fajla ("encoded_text.bin") svega 1KB, međutim ovo je relativno mali fajl koji se kompresuje pa je i stepen kompresije relativno mali.

Name	Date modified	Type	Size
 code_table	5/6/2024 4:45 PM	Text Document	6 KB
 encoded_text	5/6/2024 4:45 PM	BIN File	12 KB
 Huffman_A_variation.deps	4/27/2024 8:47 PM	JSON Source File	1 KB
 Huffman_A_variation.dll	5/6/2024 4:36 PM	Application exten...	11 KB
 Huffman_A_variation	5/6/2024 4:36 PM	Application	151 KB
 Huffman_A_variation.pdb	5/6/2024 4:36 PM	Program Debug D...	13 KB
 Huffman_A_variation.runtimeconfig	4/27/2024 8:47 PM	JSON Source File	1 KB
 initial_code_table	5/6/2024 4:42 PM	Text Document	6 KB
 input	5/6/2024 12:19 PM	Text Document	13 KB

Zaključak

A varijanta Huffman-ovog kodiranja se zasniva na korišćenju tabele umesto binarnog stabla, što omogućava lakšu implementaciju i brže izvršavanje same kompresije podataka zato što, umesto obilaženja stabla i vršenja operacija transformacije nad njim (koje znaju biti veoma skupe), jednostavno vršimo prolazak kroz tabelu i njeno sortiranje. Međutim, mana ove varijante algoritma je ta što ne vrši dovoljno efikasnu kompresiju, s obzirom da kodovi nisu zasnovani na stvarnim šansama pojavljivanja već su na početku nasumično dodeljeni.

Uzimajući u obzir prednosti i mane ovog algoritma, on ipak predstavlja značajnu tehnologiju za kompresiju podataka, sa potencijalom za primenu u različitim oblastima, uključujući prenos podataka u realnom vremenu i u drugim aplikacijama gde je brzina i efikasno prilagođavanje ključno.

Reference

1. <https://drive.google.com/file/d/18qt5ZleRwHOOimXVtKVaxSE61M1h0Di6/view>
2. <https://www.slideshare.net/slideshow/huffman-coding-algorithm-presentation-203431702/203431702>
3. <https://www.geeksforgeeks.org/adaptive-huffman-coding-and-decoding/>
4. https://en.wikipedia.org/wiki/Adaptive_Huffman_coding