Algoritmi za kompresiju podataka

Valenta, Luka

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Luka Valenta

ALGORITMI ZA KOMPRESIJU PODATAKA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, rujan, 2019.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Moderna računala pohranjuju i obrađuju velike količine podataka, ali neki dijelovi tih podataka su suvišni. *Kompresija podataka* je proces koji smanjuje veličinu podataka tako da uklanja suvišne dijelove. Kompresija podataka je danas svuda. Slike na internetu i na našim uređajima su stisnute kompresijom i spremljene, najčešće u JPEG, PNG ili GIF formatima. Bez kompresije, jedna snimka filma u visokoj kvaliteti zauzela bi velik dio tvrdog diska prosječnog računala. Zahvaljujući tome što je spremljena u smanjenom formatu koji se dobiva kompresijom, ona stane na jedan prosječni USB ili CD.

U nastavku, koristit ćemo generički termin *poruka* za objekte koje želimo smanjiti, a oni mogu biti dokumenti (datoteke) ili stvarne poruke. Svaki zadatak kompresije podataka sastoji se od dva dijela, algoritma za kompresiju podataka i algoritma za dekompresiju podataka. *Algoritam za kompresiju* kao ulazni podatak prima poruku i iz nje generira prostorno smanjenu reprezentaciju poruke. *Algoritam za dekompresiju* kao ulazni podatak prima poruku smanjenu algoritmom za kompresiju i rekonstruira originalnu poruku ili neku aproksimaciju originalne poruke. Ti algoritmi su usko povezani, jer moraju poznavati istu prostorno smanjenu reprezentaciju podataka.

Algoritme za kompresiju podataka možemo klasificirati na nekoliko načina. Jedan od najvažnijih kriterija klasifikacije je da li algoritam za kompresiju uklanja neke dijelove podataka, koje pripadni algoritam za dekompresiju ne može rekonstruirati pri dekompresiji. Drugim riječima, da li algoritam za dekompresiju uvijek rekonstruira originalnu poruku ili to može biti i aproksimacija originalne poruke. Ako je rekonstrukcija uvijek ista kao i originalna poruka, radi se o tzv. *Lossless algoritmu* (algoritmu bez gubitka), a algoritam kod kojeg ima gubitaka zove se *Lossy algoritam*.

Lossy algoritmi se koriste kada nije nužno da podaci, nakon dekompresije, točno odgovaraju originalnim podacima. Konkretno, lossy kompresija se koristi kod kompresije slika i videa. Koristi se činjenica da, ukoliko je primatelj slike ili videa čovjek, postoje neke informacije čiji gubitak čovjek neće primijetiti. Primjerice, čovjeku je teško primijetiti male varijacije u boji pa piksele slične boje možemo spremiti kao da su svi potpuno iste boje. Lossy kompresija se koristi u ovakvim slučajevima, jer tipično daje puno bolje omjere kompresije od lossless algoritama. Međutim, za neke tipove podataka, iz tehničkih ili pravnih razloga, nije moguće koristiti lossy kompresiju. Na primjer, u mnogim zem-

ljama, po zakonu se lossy kompresija ne smije koristiti za slike korištene u medicini. Kod teksta se, također, koristi lossless dekompresija. Ako bismo dopustili da se neka slova u originalnom tekstu mogu promijeniti, čovjek koji ga čita lako bi primijetio greške.

Ne postoji lossless algoritam koji bi mogao smanjiti veličinu svake poruke iz nekog skupa mogućih poruka. Razlog tome je što ako algoritam skrati duljinu neke poruke, on nužno mora produljiti neku drugu poruku. Algoritmi za kompresiju postižu dobre omjere kompresije tako što pretpostavljaju da vjerojatnosti različitih poruka variraju i to koriste kako bi porukama s većom vjerojatnosti pojavljivanja pridružili kraće reprezentacije, a onima s manjom vjerojatnosti dulje. Iako ukupna duljina svih mogućih poruka nije manja, duljina promatrane poruke će u prosjeku biti manja.

Algoritme za kompresiju, također, dijelimo na *univerzalne* i na algoritme za *kompresiju specifičnih formata*. Kažemo da je algoritam za kompresiju podataka univerzalan ako dobro radi na podacima koji mogu biti aproksimirani kao izlaz neke generičke klase poruka.

Kod algoritama za kompresiju gledamo dvije važne komponente: model i koder. *Model* opisuje vjerojatnosnu distribuciju poruka na temelju poznavanja strukture ulaznih nizova poruka. *Koder* koristi vjerojatnosnu distribuciju generiranu u modelu da bi generirao kodove. Koder generira kodove tako da smanji duljine poruka s većom vjerojatnosti, a produlji poruke s manjom vjerojatnosti. Postoji puno različitih dizajna model komponente i puno različitih razina složenosti modela, ali razlike među koder komponentama su obično male. Većina algoritama za kompresiju koji su danas u upotrebi koristi Huffmanovo ili aritmetičko kodiranje. Ipak, podjela na model i koder nije jasno definirana u svim algoritmima za kompresiju podataka.

U prvom poglavlju proći ćemo osnove informacijske teorije. Informacijsku teoriju možemo smatrati ljepilom između modela i kodera u algoritmima za kompresiju. Ona nam daje teoriju o tome kako su vjerojatnosti povezane sa sadržajem informacija i duljinom kodova. U drugom poglavlju promotrit ćemo vezu između informacijske teorije i kodera, kroz prefiksno, Huffmanovo i aritmetičko kodiranje. U poglavljima tri, četiri i pet opisat ćemo neke algoritme za univerzalnu kompresiju podataka, a u posljednjem, šestom poglavlju uspoređuje se veličina i kvaliteta slika iz odabranih datasetova spremljenih u PNG i JPEG formatima.

Poglavlje 1

Teorija informacija

1.1 Entropija

Začetnikom teorije informacija smatra se Claude Shannon, koji je 1948. godine u članku *"A Mathematical Theory of Communication"* uveo pojam entropije informacija i dao osnovne postavke za teoriju informacija [1]. On je definiciju entropije preuzeo iz statističke fizike. U statističkoj fizici entropija predstavlja nasumičnost ili neuniformnost sistema. Pretpostavlja se da postoji skup mogućih stanja u kojima sistem može biti i da je, u bilo kojem datom trenutku, dana vjerojatnosna distribucija sistema na tom skupu stanja. *Entropija* je definirana kao

$$H(S) = \sum_{s \in S} \mathbb{P}(s) \log_2 \frac{1}{\mathbb{P}(s)},$$

gdje je S skup mogućih stanja, a $\mathbb{P}(s)$ je vjerojatnost stanja $s \in S$. Iz ove definicija slijedi da uniformnija vjerojatnosna distribucija rezultira većom entropijom, a neujednačenija vjerojatnosna distribucija vodi do manje entropije. Za ilustraciju navedenog svojstva pogledajmo entropije nekoliko skupova od četiri elementa, koji imaju različite vjerojatnosne distribucije. Brojeve zaokružujemo na šest decimala.

$$S_1 = \{s_1^1, s_2^1, s_3^1, s_4^1\}, \quad \mathbb{P}(s_1^1) = \mathbb{P}(s_2^1) = \mathbb{P}(s_3^1) = \mathbb{P}(s_4^1) = \frac{1}{4}.$$

$$S_2 = \{s_1^2, s_2^2, s_3^2, s_4^2\}, \quad \mathbb{P}(s_1^2) = \mathbb{P}(s_2^2) = \frac{3}{8}, \;\; \mathbb{P}(s_3^2) = \mathbb{P}(s_4^2) = \frac{1}{8}.$$

$$S_3 = \{s_1^3, s_2^3, s_3^3, s_4^3\}, \quad \mathbb{P}(s_1^3) = \frac{5}{8}, \;\; \mathbb{P}(s_2^3) = \mathbb{P}(s_3^3) = \mathbb{P}(s_4^3) = \frac{1}{8}.$$

U sva tri slučaja, entropija je dana izrazom

$$H(S_i) = \sum_{j=1}^{4} \mathbb{P}(s_j^i) \log_2 \frac{1}{\mathbb{P}(s_j^i)},$$

odakle dobivamo

$$H(S_1) = 4 \cdot \frac{1}{4} \cdot \log_2(4) = 2,$$

$$H(S_2) = 2 \cdot \frac{3}{8} \cdot \log_2 \frac{8}{3} + 2 \cdot \frac{1}{8} \cdot \log_2 8 = 1.811278,$$

$$H(S_3) = \frac{5}{8} \cdot \log_2 \frac{8}{5} + 3 \cdot \frac{1}{8} \cdot \log_2 8 = 1.548795.$$

U primjeru primjećujemo da entropija pada kako vjerojatnosne distribucije na skupovima S_i postaju manje uniformne.

Za potrebe informacijske teorije Shannon je stanja zamijenio porukama pa je S skup mogućih poruka, a $\mathbb{P}(S)$ vjerojatnost poruke $s \in S$. Također, Shannon je definirao pojam *vlastite informacije* ili *informacije o sebi* (eng. *self information*) poruke s, kao

$$i(s) = \log_2 \frac{1}{\mathbb{P}(s)}.$$

Vlastita informacija poruke predstavlja broj bitova informacije sadržan u poruci i upućuje na to koliko bitova bismo trebali koristiti za kodiranje te poruke. Vlastite informacije poruka većih vjerojatnosti bit će manje od onih poruka veće vjerojatnosti pa će za kodiranje poruka veće vjerojatnosti biti potrebno manje bitova. Na primjer, vjerojatnost da je izjava "sljedeći tjedan će u Zagrebu padati kiša" istinita veća je od vjerojatnosti da je izjava "idući ponedjeljak će u Zagrebu padati kiša" istinita. Prva izjava je stoga i manje informativna. Uočavamo da možemo zapisati

$$H(S) = \sum_{s \in S} \mathbb{P}(s) \cdot \log_2 \frac{1}{\mathbb{P}(s)} = \sum_{s \in S} \mathbb{P}(s) \cdot i(s)$$

pa je entropija jednaka težinskom prosjeku vlastitih informacija svih poruke, gdje težina odgovara vjerojatnosti poruke. Radi se o težinskom prosjeku zato što je $\sum_{s \in S} \mathbb{P}(s) = 1$. Dakle, entropija odgovara prosječnom broju bitova informacije sadržanom u poruci nasumično odabranoj iz skupa, s time da je vjerojatnost odabira poruke s jednaka $\mathbb{P}(s)$.

Pojasnimo još zašto je za definiciju vlastite informacije odabrana funkcija $\log_2(1/\mathbb{P}(s))$. Svojstva koja funkcija i mora zadovoljavati da bi bila adekvatna za definiciju vlastite informacije su:

1. Neka je S skup od $n = 2^i$ poruka, od kojih je svaka vjerojatnosti $1/n$. Ukoliko su sve poruke $s \in S$ jednake duljine, tada je $\log_2 n$ bitova potrebno da se poruka kodira u binarnom zapisu. Dakle, u tom slučaju očekujemo da je
$$i(s) = \log_2 \frac{1}{\mathbb{P}(s)} = \log_2 n.$$
2. Vlastita informacija dvije nezavisne poruke (poslane jedne za drugom) mora biti jednaka sumi vlastitih informacija svake poruke. Dakle, očekujemo da funkcija ima svojstvo $i(ab) = i(a) + i(b)$, gdje su a i b nezavisne poruke.

Očito je zadovoljeno 1. svojstvo. Pokazujemo da zadovoljeno i drugo. Budući da su poruke a i b nezavisne, vjerojatnost slanja jedne za drugom je $\mathbb{P}(ab) = \mathbb{P}(a) \cdot \mathbb{P}(b)$. Onda je
$$i(ab) = \log_2 \frac{1}{\mathbb{P}(ab)} = \log_2 \frac{1}{\mathbb{P}(a) \cdot \mathbb{P}(b)} = \log_2 \frac{1}{\mathbb{P}(a)} + \log_2 \frac{1}{\mathbb{P}(a)} = i(a) + i(b).$$
Dakle, $\log_2(1/\mathbb{P}(s))$ zadovoljava nužna svojstva koja očekujemo od funkcije za definiciju vlastite informacije, a uz to je i "najjednostavnija" takva funkcija koju znamo, pa je ona odabrana za definiciju.

1.2 Uvjetna entropija i Markovljevi lanci

Vjerojatnosti događaja, odnosno, poruka u teoriji informacija, često ovise o kontekstu u kojem se događaju. Na primjer, vjerojatnost događaja da je u Zagrebu preko 25°C bit će različita ovisno o tome koji je mjesec u godini. Slično, korištenjem prethodnih slika u video zapisu možemo puno bolje predvidjeti kakvi će biti pikseli u sljedećoj slici video zapisa, nego što bismo da svaku sliku kodiramo neovisno o prethodnima. Korištenjem informacija u kontekstu možemo poboljšati preciznost vjerojatnosti na koje se oslanjamo pri kodiranju poruka i tako smanjiti entropiju. Uvjetna vjerojatnost događaja e uz uvjet (kontekst) c definirana je kao
$$\mathbb{P}(e|c) = \frac{\mathbb{P}(e \cap c)}{\mathbb{P}(c)}.$$
Za skup svih mogućih konteksta C vrijedi
$$\mathbb{P}(e) = \sum_{c \in C} \mathbb{P}(e|c)\mathbb{P}(c),$$
zato što je C potpun skup događaja, odnosno, ne postoji kontekst koji nije u C i svaka dva konteksta u C su disjunktna. Na temelju uvjetnih vjerojatnosti, definiramo *uvjetnu vlastitu informaciju* događaja e u konteksu c kao
$$i(e|c) = \log_2 \frac{1}{\mathbb{P}(e|c)}.$$

Uvjetna vlastita informacija $i(e|c)$ ne mora biti ista kao $i(c)$. Ako se poslužimo prethodnim primjerom, informacija da je u Zagrebu temperatura preko 25°C kaže nam više, nego ista informacija u kontekstu da je sada kolovoz.

Sada definiramo uvjetnu entropiju kao prosjek uvjetnih vlastitih informacija, po svim porukama i po svim kontekstima. Za skup poruka S i skup konteksta C, *uvjetna entropija* je

$$H(S|C) = \sum_{c\in C} \mathbb{P}(c) \sum_{s\in S} \mathbb{P}(s|c) \log_2 \frac{1}{\mathbb{P}(s|c)}.$$

Teorem 1.2.1. *Neka je S skup poruka s nekom vjerojatnosnom distribucijom, a C skup konteksta. Tada vrijedi $H(S|C) \leq H(S)$, pri čemu vrijedi jednakost ako i samo ako je S neovisan o C.*

Dokaz. Računamo razliku $H(S|C) - H(S)$, uz ocjenu odozgo na odgovarajućem mjestu.

$$\begin{aligned}
H(S|C) - H(S) &= \sum_{c\in C} \mathbb{P}(c) \sum_{s\in S} \mathbb{P}(s|c) \log_2 \frac{1}{\mathbb{P}(s|c)} - \sum_{s\in S} \mathbb{P}(s) \log_2 \frac{1}{\mathbb{P}(s)} \\
&= \sum_{c\in C} \sum_{s\in S} \mathbb{P}(c) \frac{\mathbb{P}(s\cap c)}{\mathbb{P}(c)} (-\log_2 \mathbb{P}(s|c)) - \sum_{c\in C} \mathbb{P}(c) \sum_{s\in S} \mathbb{P}(s)(-\log_2(\mathbb{P}(s))) \\
&= \sum_{c\in C} \sum_{s\in S} \mathbb{P}(s\cap c) \log_2(\mathbb{P}(s)) - \sum_{c\in C} \sum_{s\in S} \mathbb{P}(s\cap c) \log_2(\mathbb{P}(s|c)) \\
&= \sum_{c\in C} \sum_{s\in S} \mathbb{P}(s\cap c) \log_2 \frac{\mathbb{P}(s)}{\mathbb{P}(s|c)} \\
&= \sum_{c\in C} \sum_{s\in S} \mathbb{P}(s\cap c) \log_2 \frac{\mathbb{P}(s)\mathbb{P}(c)}{\mathbb{P}(s\cap c)} \\
&\leq \log_2 \sum_{c\in C} \sum_{s\in S} \mathbb{P}(s\cap c) \frac{\mathbb{P}(s)\mathbb{P}(c)}{\mathbb{P}(s\cap c)} \\
&= \log_2 \sum_{c\in C} \sum_{s\in S} \mathbb{P}(s)\mathbb{P}(c) \\
&= \log_2 \sum_{c\in C} \left(\mathbb{P}(c) \sum_{s\in S} \mathbb{P}(s) \right) = \log_2 \sum_{c\in C} \mathbb{P}(c) = \log_2 1 = 0.
\end{aligned}$$

U šestoj liniji, gdje je nejednakost, koristimo Jensenovu nejednakost i činjenicu da je $\log_2$ konkavna funkcija. Slijedi $H(S|C) \leq H(S)$. Dodatno, u liniji gdje je nejednakost, jednakost vrijedi ako i samo ako je argument od $\log_2$ konstanta, odnosno, ako $\mathbb{P}(s|c) = \mathbb{P}(s)$, za svaki $s \in S$ i za svaki $c \in C$. To pak vrijedi ako i samo ako je skup poruka S neovisan o skupu konteksta C. □

Prethodni teorem nam zapravo kaže da poznavanje konteksta može samo smanjiti entropiju.

Shannon je entropiju definirao u terminima informacijskih izvora. *Informacijski izvor* generira beskonačan niz poruka $X_k, k \in \langle -\infty, \infty \rangle$, iz fiksnog skupa poruka S. Ako je vjerojatnost svake poruke neovisna o prethodnim porukama onda se sistem naziva *neovisan i identično distribuiran izvor*, a entropija tog izvora se naziva *bezuvjeta entropija* ili *entropija prvog reda*. Entropija koju smo definirali u odjeljku 1.1 je entropija prvog reda i nju ćemo nastaviti kratko zvati entropijom.

Druga vrsta izvora poruka je Markovljev lanac s diskretnim vremenom. Niz slijedi Markovljev model k-tog reda ako vjerojatnost svake poruke ovisi samo o prethodnih k poruka. Entropija Markovljevog lanca s diskretnim vremenom je definirana uvjetnom entropijom, koja se temelji na uvjetnim vjerojatnostima $\mathbb{P}(x_n|x_{n-1}, \ldots, x_{n-k})$.

Shannon je definirao i pojam entropije izvora za proizvoljan izvor. Neka je A^n skup svih nizova duljine n iz abecede A. Tada je *normalizirana entropija n-tog reda* definirana kao

$$H_n = \frac{1}{n} \sum_{X \in A^n} \mathbb{P}(X) \log \frac{1}{\mathbb{P}(X)}.$$

Budući da je ona normalizirana dijeljenjem s n, ona predstavlja informaciju po znaku. *Entropija izvora* se onda definira kao

$$H = \lim_{n \to \infty} H_n.$$

Entropiju proizvoljnog izvora teško je odrediti gledajući samo izlaz izvora, jer za precizno računanje vjerojatnosti moramo gledati jako duge nizove znakova.

Poglavlje 2

Vjerojatnosno kodiranje

U praksi se vjerojatnosti koriste za dijelove veće poruke, a ne za cijele poruke. Na primjer, poruka neće biti cijela slika, nego pojedini pikseli na slici. Svaki od tih manjih dijelova smatrat ćemo jednom porukom, a koristit ćemo naziv *niz poruka* za veću poruku, koja se sastoji od tih manjih dijelova. Svaka manja poruka može dolaziti iz različite vjerojatnosne distribucije i može biti različitog tipa.

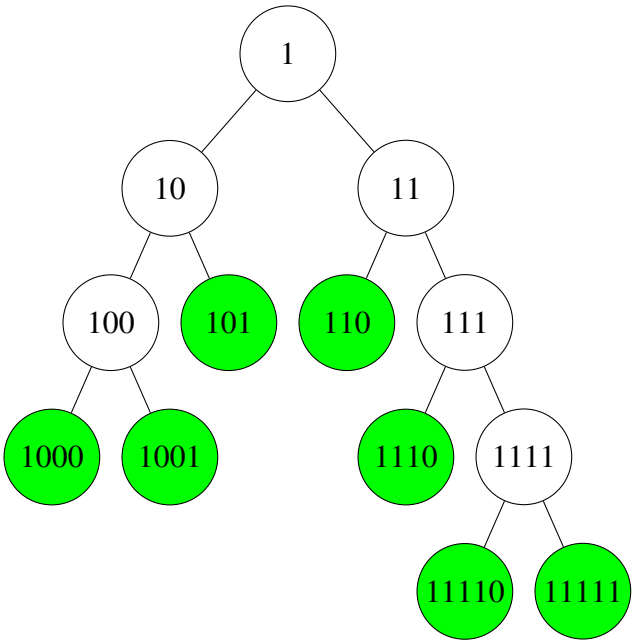
Neki algoritmi svakoj poruci dodijele jedinstven kod, a neki spajaju više poruka i kodiraju ih zajedno. Huffmanovi kodovi, koji su vrsta prefiksnih kodova, svakoj poruci dodijele jedinstven kod, dok aritmetički kodovi kodiraju više poruka zajedno. Zahvaljujući tome, aritmetički kodovi postižu bolju kompresiju, ali koder nekad mora odgoditi slanje poruke, jer mora biti kodirana zajedno s nekim drugim porukama, prije nego što se može poslati.

2.1 Prefiksno kodiranje

Kod C za skup poruka S je preslikavanje sa skupa poruka u skup konačnih nizova bitova, koje ćemo nazivati *kodnim riječima*. Kodove ćemo zapisivati u obliku

$$C = \{(s_1, w_1), (s_2, w_2), \ldots, (s_m, w_m)\},$$

gdje je s_i poruka, a w_i pripadna kodna riječ. Prilikom kodiranja želimo porukama s većom vjerojatnosti pridružiti kraće kodne riječi, a manje vjerojatnima duže. Kod takvih kodnih riječi javlja se problem da, kada ih šaljemo jednu za drugom, može biti teško ili nemoguće odrediti gdje jedna kodna riječ završava, a druga počinje. Kako bismo rješili taj problem, možemo dodati poseban simbol između kodnih riječi ili, prije svake kodne riječi, poslati njezinu duljinu. Međutim, ta rješenja iziskuju slanje dodatnih podataka. Kako bismo to izbjegli, koristimo jedinstveno dekodirajuće kodove. *Jedinstveno dekodirajući kodovi* su kodovi kod kojih uvijek možemo jedinstveno dešifrirati niz bitova u kodnu riječ.



Slika 2.1: Primjer binarnog stabla prefiksnog koda

Prefiksni kod je posebna vrsta jedinstveno dekodirajućih kodova kod kojeg ni jedan niz bitova nije prefiks drugog, na primjer

$$C = \{(a, 101), (b, 1000), (c, 1001), (d, 110), (e, 1110), (f, 11110), (g, 11111)\}.$$

Dodatna prednost prefiknih kodova je što možemo dešifrirati svaku poruku bez da vidimo početak sljedeće poruke. To je korisno kada šaljemo poruke različitih tipova ili kada jedna poruka specificira tip sljedeće poruke pa je nužno da se ta poruka dekodira, kako bismo saznali tip sljedeće poruke, koji nam pak treba kako bismo sljedeću poruku dekodirali.

Na prefiksni kod možemo gledati kao na binarno stablo kod kojeg je svaka poruka list u stablu. Kod za poruku dobivamo slijedeći put od korijena do lista — dodajući 0 svaki put kada je uzeta lijeva grana, odnosno, 1 svaki put kada je uzeta desna grana (v. sliku 2.1).

Općenito se prefiksni kodovi mogu koristiti i s abecedama koje nisu binarne. U slučaju abecede s n znakova, u stablu bi čvorovi imali maksimalno n djece. Mi ćemo promatrati samo binarne prefiksne kodove.

Za danu vjerojatnosnu distribuciju na skupu poruka i pridruženi kod C promjenjive duljine, definiramo *prosječnu duljinu koda* kao

$$l_a(C) = \sum_{(s,w)\in C} \mathbb{P}(s)l(w),$$

gdje je $l(w)$ duljina kodne riječi w. Kažemo da je prefiksni kod *C optimalan prefiksni kod*, ako ne postoji prefiksni kod za danu vjerojatnosnu distribuciju koji ima manju prosječnu duljinu.

Odnos s entropijom

Sljedeće dvije leme daju donju i gornju među za prosječnu duljinu koda. Donja međa odgovara entropiji, a gornja je za jedan veća od entropije, s tim da se gornja odnosi na optimalne prefiksne kodove.

Lema 2.1.1. *Za svaki skup poruka S s vjerojatnosnom distribucijom i pridruženi jedinstveno dekodirajući kod C, vrijedi* $H(S) \leq l_a(C)$.

Dokaz. Dokaz leme možete pronaći u [6]. □

Lema 2.1.2. *Za svaki skup poruka S s vjerojatnosnom distribucijom i pridruženi optimalan prefiksni kod C, vrijedi* $l_a(C) \leq H(S) + 1$.

Dokaz. Dokaz leme možete pronaći u [6]. □

Sljedeći teorem pokazuje da, kod optimalnih prefiksnih kodova, veće vjerojatnosti nikad ne odgovaraju duljim kodovima.

Teorem 2.1.3. *Neka je* $C = \{(s_1, w_1), (s_2, w_2), \ldots, (s_m, w_m)\}$ *optimalan prefiksni kod za vjerojatnosti* $\{\mathbb{P}(s_1), \mathbb{P}(s_2), \ldots, \mathbb{P}(s_m)\}$. *Ako vrijedi* $\mathbb{P}(s_i) > \mathbb{P}(s_j)$ *tada vrijedi* $l(w_i) \leq l(w_j)$.

Dokaz. Dokaz teorema možete pronaći u [6]. □

2.2 Huffmanovo kodiranje

David Huffman je 1950. godine, još kao student na MIT-u, razvio algoritam za kodiranje koji je postao poznat kao *Huffmanov algoritam*. On je danas jedna od najčešće korištenih komponenti u algorimima za kodiranje pa se tako koristi u GZIP-u, JPEG-u i mnogim drugim alatima.

Huffmanov algoritam je vrlo jednostavan i najlakše ga je opisati u terminima stabala. Za zadani skup $S = \{s_1, s_2, \ldots, s_n\}$ s pripadnom vjerojatnosnom distribucijom, Huffmanov algoritam generira stablo prefiksnog koda u dva koraka. Prvi korak je inicijalizacija, a drugi je glavna petlja.

1. Generiraj šumu stabala, po jedno stablo za svaku poruku. Svako stablo se sastoji od samo jednog vrha s težinom $w_i = \mathbb{P}(s_i)$.

2. Ponavljaj dok ne ostane samo jedno stablo:

   a) Odaberi dva stabla s najmanjim težinama u korijenima. Te težine označimo s w_1 i w_2.

   b) Spoji ta dva stabla u jedno stablo tako da je korijen tog stabla čvor s težinom $w_1 + w_2$, a stabla koja želimo spojiti su djeca tog čvora. Nije bitno koje stablo će biti lijevo, a koje desno dijete, ali konvencija je da stablo s manjom težinom bude lijevo dijete, ukoliko su stabla različitih težina.

Huffmanovi kodovi su prefiksni kodovi generirani *Huffmanovim algoritmom*. Za kod duljine n, algoritam će imati $n-1$ koraka, jer svako potpuno binarno stablo s n listova ima $n-1$ čvorova koji nisu listovi, a svaki korak stvara jedan čvor koji nije list. Ako koristimo prioritetni red, kod kojeg je vremenska složenost ubacivanja u red i nalaženja minimuma $O(\log n)$, vremenska složenost cijelog algoritma bit će $O(n \log n)$.

Lema 2.2.1. *Huffmanov algoritam generira optimalan prefiksni kod.*

Dokaz. Dokaz je indukcijom po broju poruka u kodu.

Baza: Prefiksni kod jedne poruke je jedinstven pa mora biti optimalan.

Pretpostavka: Huffmanov algoritam generira optimalan prefiksni kod za sve vjerojatnosne distribucije n poruka.

Korak: Neka je S skup poruka veličine $n+1$ s pripadnom vjerojatnosnom distribucijom i neka je T stablo dobiveno iz tog skupa Huffmanovim algoritmom. Neka su x i y dva čvora s najmanjim vjerojatnostima u stablu T. Zbog dizajna algoritma, oni moraju imati istog roditelja. Promotrimo stablo T', dobiveno iz T tako da čvorove x i y zamijenimo njihovim roditeljem, nazovimo ga z. Njegova vjerojatnost je $\mathbb{P}(z) = \mathbb{P}(x) + \mathbb{P}(y)$. Dubinu od z označimo s d. Onda je

$$\begin{aligned} l_a(T) &= l_a(T') + \mathbb{P}(x) \cdot (d+1) + \mathbb{P}(y) \cdot (d+1) - \mathbb{P}(z) \cdot d \\ &= l_a(T') + \mathbb{P}(x) \cdot (d+1) + \mathbb{P}(y) \cdot (d+1) - \mathbb{P}(x) \cdot d - \mathbb{P}(y) \cdot d \\ &= l_a(T') + \mathbb{P}(x) + \mathbb{P}(y). \end{aligned}$$

Tvrdimo da postoji optimalan kod za S, u kojem dva čvora s najmanjim vjerojatnostima imaju istog roditelja u stablu. Po Kraft-McMillanovoj nejednakosti, iskazanoj u [6], znamo da su ta dva čvora na najnižoj razini u stablu. Također, možemo zamijeniti bilo koje čvorove na najnižoj razini u stablu, bez utjecaja na prosječnu duljinu koda, jer svi ti kodovi imaju istu duljinu. Stoga, bez smanjenja općenitosti, možemo pretpostaviti da dva čvora s najmanjim vjerojatnostima imaju istog roditelja.

Postoji optimalan kod u čijem stablu x i y imaju istog roditelja pa, gdje god stavimo te čvorove, oni će dodati $\mathbb{P}(x) + \mathbb{P}(y)$ na prosječnu duljinu bilo kojeg prefiksnog stabla na S, u kojem su x i y zamijenjeni sa svojim roditeljem z. Po pretpostavci indukcije, $l_a(T')$

je minimiziran, jer T' ima n listova i generiran je Huffmanovim algoritmom, pa je onda i $l_a(T)$ minimiziran i T je optimalan. □

Iz optimalnosti Huffmanovog kodiranja i lema 2.1.1 i 2.1.2 slijedi da za bilo koju vjerojatnosnu distribuciju na skupu poruka S i pripadni Huffmanov kod C vrijedi

$$H(S) \leq l_a(C) \leq H(S) + 1.$$

Grupiranje poruka

Prefiksni kodovi mogu biti vrlo neefikasni. Ako je $H(S)$ puno manji od 1, onda $H(S) + 1$ može biti znatno veći od $H(S)$, relativno obzirom na $H(S)$. Jedan mogući način da se poboljša algoritam je da se više poruka grupira u jednu i tako smanji "posao" koji se mora obaviti za svaku poruku. To je lako ako su sve poruke iz iste vjerojatnosne distribucije. Ako postoji n mogućih poruka i grupiramo poruke u grupe od dvije, onda možemo generirati vjerojatnosnu distribuciju za svih n^2 mogućih parova, množenjem vjerojatnosti poruka u paru. Primijetimo da parovi (a, b) i (b, a) imaju istu vjerojatnost. Ako gledamo kombinacije poruka u $n \times n$ matrici, radit će se o simetričnoj matrici. To znači da će biti najviše $\sum_{i=1}^{n} i = n(n + 1)/2$ različitih vjerojatnosti. Možemo generirati Huffmanov kod za tu novu vjerojatnosnu distribuciju i koristiti ga za kodiranje dvije po dvije poruke. Grupiranjem k poruka, dodatan "posao" koji se mora odraditi u Huffmanovom algoritmu može se smanjiti s 1 bita po poruci, na $1/k$ bitova po poruci.

Problemi s ovakvim načinom optimizacije su što poruke često nisu iz iste vjerojatnosne distribucije i što spajanje poruka može biti skupo, zbog svih mogućih kombinacija vjerojatnosti koje se moraju generirati.

Huffmanovi kodovi s minimalnom varijancom

Za neke primjene može biti korisno smanjiti varijancu u duljini koda. *Varijanca duljine koda* je definirana kao

$$\sum_{c \in C} \mathbb{P}(c) \left(l(c) - l_a(C)\right)^2 .$$

S manjom varijancom može biti lakše postići konstantnu brzinu slanja znakova ili smanjiti veličinu spremnika za skupljanje podataka prije slanja.

Kako bismo minimizirali varijancu kodova dobivenih Huffmanovim algoritmom, dovoljno je napraviti jednu modifikaciju. Kada postoji više stabala jednake težine koje možemo spojiti, uvijek biramo čvor koji je stvoren najranije u algoritmu.

2.3 Aritmetičko kodiranje

Aritmetičko kodiranje je tehnika kodiranja koja omogućuje da se informacije iz poruka u nizu kodiraju tako da dijele iste bitove. Takav način kodiranja omogućuje da se ukupan broj bitova asimptotski približava sumi vlastitih informacija poruka u nizu. To je posebno značajno kada su vjerojatnosti velike.

Ideja aritmetičkog kodiranja je predstaviti svaki mogući niz od n poruka zasebnim (disjunktnim) intervalom koji je podskup od $[0, 1]$. Nizu poruka $s_1, \ldots, s_n$ s vjerojatnostima $\mathbb{P}(s_1), \mathbb{P}(s_2), \ldots, \mathbb{P}(s_n)$, algoritam će pridružiti interval veličine $\sum_{i=1}^{n} \mathbb{P}(s_i)$, koji se odredi tako da počnemo s intervalom $[0, 1\rangle$, a zatim ga sužavamo za faktor $\mathbb{P}(s_i)$ na svakoj sljedećoj poruci s_i. Možemo odrediti gornju među za broj bitova potreban da se jedinstveno odredi interval veličine s i iskoristiti tu među za usporedbu duljine reprezentacije s vlastitom informacijom tih poruka.

Pretpostavljamo da dekoder zna kada je niz poruka završen, tako što zna koja je duljina niza ili tako što nakon niza slijedi poseban znak za kraj dokumenta. Ova pretpostavka je implicitno postojala i kod Huffmanovih kodova, jer dekoder mora znati kada je niz poruka gotov.

Vjerojatnosne distribucije nekog skupa poruka označavamo s $\mathbb{P}(1), \ldots, \mathbb{P}(m)$. Za takvu vjerojatnosnu distribuciju definiramo *akumuliranu vjerojatnost* kao

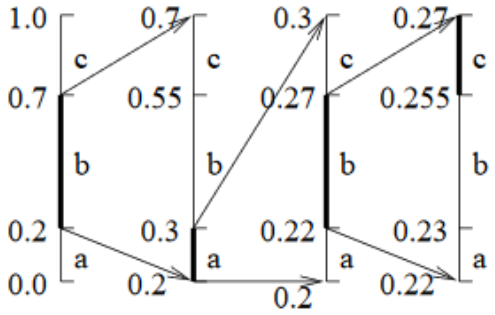
$$f(j) = \sum_{i=1}^{j-1} \mathbb{P}(i), \quad j = 1, \ldots, m.$$

Kod nizova poruka, gdje svaka može biti iz različite vjerojatnosne distribucije, vjerojatnosnu distribuciju i-te poruke označavat ćemo s $\mathbb{P}_i(1), \ldots \mathbb{P}_i(m_i)$, a akumulirane vjerojatnosti s $f_i(1), \ldots, f_i(m_i)$. Za konkretan niz vrijednosti poruka, indeks vrijednosti i-te poruke označavamo s v_i. Koristimo skraćene oznake p_i za $\mathbb{P}_i(v_i)$ i f_i za $f_i(v_i)$.

Aritmetičko kodiranje nizu poruka pridružuje interval koristeći sljedeće rekurzije:

$$\begin{aligned} l_i &= \begin{cases} f_i, & i = 1, \\ l_{i-1} + f_i \cdot s_{i-1}, & 1 < i \leq n, \end{cases} \\ s_i &= \begin{cases} p_i, & i = 1, \\ s_{i-i} \cdot p_i, & 1 < i \leq n, \end{cases} \end{aligned} \tag{2.1}$$

gdje je l_n lijeva granica intervala, a s_n veličina intervala, tj. interval je oblika $[l_n, l_n + s_n\rangle$. Pretpostavljamo da se radi o poluotvorenom intervalu, gdje je lijeva granica uključena u interval, a desna nije. Na slici 2.2 vidi se ilustracija generiranja intervala za niz poruka *babc*. Lako se pokaže da su intervali generirani jednadžbama (2.1) disjunktni, za sve jedinstvene nizove poruka duljine n. Zbog toga, na temelju intervala možemo u dekoderu jedinstveno odrediti niz poruka. Štoviše, dovoljno je specificirati bilo koji broj unutar tog intervala, jer



Slika 2.2: Primjer generiranja aritmetičkog koda pod pretpostavkom da su sve poruke iz iste vjerojatnosne distribucije $\mathbb{P}(a) = 0.2$, $\mathbb{P}(b) = 0.5$ i $\mathbb{P}(c) = 0.3$. Interval generiran za niz poruka *babc* je $[0.255, 0.27\rangle$. Primjer je preuzet iz [6].

su intervali disjunktni. Dekoder onda radi isto što i koder, ali u obrnutnom smjeru. On koristi interval za izbor vrijednosti poruke, a zatim ga sužava i tako dekodira niz poruka.

Postavlja se pitanje koji ćemo broj iz intervala koristiti za reprezentaciju intervala i kako ga poslati u binarnoj reprezentaciji. Realni brojevi između 0 i 1 mogu biti zapisani u binarnoj reprezentaciji na način koji se vidi iz sljedećeg primjera — realnom broju $9/16$ odgovara binarni zapis 0.1001.

$$\begin{aligned} 9/16 \geq 1/2 &\to 1 \\ 9/16 < 3/4 &\to 0 \\ 9/16 < 5/8 &\to 0 \\ 9/16 = 9/16 &\to 1. \end{aligned}$$

Međutim, ne možemo samo uzeti broj iz intervala s najkraćim zapisom u binarnoj reprezentaciji, jer skup takvih kodova nije skup prefiksnih kodova. Da bismo izbjegli taj problem, svaku kodnu riječ u binarnom zapisu tumačimo kao interval svih mogućih dopunjenja. Ključna riječ 0.100 tako predstavlja interval $[1/2, 5/8\rangle$, jer je najmanje moguće dopunjenje $0.100\overline{0} = 1/2$, a najveće $0.100\overline{1} = 5/8$ (gdje $\overline{w}$ označava beskonačno ponavljanje niza w).

Kako bismo lakše razlikovali o kojim vrstama intervala govorimo, trenutni interval niza poruka $[l_i, l_i + s_i\rangle$ nazivat ćemo *interval niza*, interval koji odgovara vjerojatnosti i-te poruke $[f_i, f_i + p_i\rangle$ nazivat ćemo *intervalom poruke*, a interval ključne riječi *kodni interval*.

Sljedeća lema kaže kako postoji direktna veza između preklapanja intervala i toga da čine prefiksni kod.

Lema 2.3.1. *Ako za kod C ne postoje dva intervala, reprezentrirana svojim binarnim kodnim rječima $w \in C$, nepraznog presjeka, onda je C prefiksni kod.*

Dokaz. Dokaz leme možete pronaći u [6]. □

Lema 2.3.2. *Za bilo koje l i s, takve da je $l, s \geq 0$ i $l + s < 1$, interval dobiven tako da se uzme binarna reprezentacija od $l + s/2$ i skrati na $\lceil -\log_2 s \rceil + 1$ bitova, je sadržan u intervalu $[l, l + s\rangle$.*

Dokaz. Dokaz leme možete pronaći u [6]. □

Sada navodimo teorem o algoritmu koji se sastoji od generiranja intervala jednadžbama (2.1) i korištenja metode skraćivanja iz leme 2.3.2. Taj algoritam ćemo, kao u [6], nazvati *RealArithCode algoritam*.

Teorem 2.3.3. *Za niz od n poruka koje imaju vlastite informacije $s_1, \ldots, s_n$, duljina aritmetičkog koda generiranog RealArithCode algoritmom je manja ili jednaka $2 + \sum_{i=1}^{n} s_i$ i taj kod neće biti prefiks niti jednog drugog niza od n poruka.*

Dokaz. Dokaz teorema možete pronaći u [6]. □

Postoji nekoliko problema s navedenim algoritmom za aritmetičko kodiranje:

1. Algoritmu je potrebna proizvoljna preciznost da bi manipulirao s l i s, koji su realni brojevi, posebno kad intervali postanu jako mali.

2. Koder ne može poslati niti jedan bit, sve dok ne kodira cijelu poruku.

Jedan od načina da se riješi prvi problem je korištenje implementacije s cijelim brojevima. Ona je manje efikasna, ali je preciznost fiksna. Više o implementaciji s cijelim brojevima možete pronaći u [6].

Prvi način na koji možemo umanjiti, ali ne i riješiti drugi problem je da šaljemo bit po bit, 0 ili 1, kada interval padne unutar donje ili gornje polovice. Međutim, to ne garantira redovito slanje podataka. U najgorem slučaju, još uvijek ćemo morati čekati da se cijeli niz poruka kodira, ako trenutni interval sadrži 0.5 sve do kraja kodiranja. Drugi pristup je da se niz poruka podijeli u blokove fiksne veličine i da se aritmetičko kodiranje koristi na svakom bloku posebno.

Poglavlje 3

Primjena vjerojatnosnog kodiranja

Kako bismo koderom kodirali podatke, potreban nam je model na temelju kojeg ćemo generirati vjerojatnosti. Dekoder koristi isti taj model kako bi dekodirao kodirani niz poruka. Model može biti:

- *Statičan nad svim nizovima poruka* — tada možemo jednostavno zapisati sve vjerojatnosti u koder i dekoder.
- *Statičan nad jednim konkretnim nizom poruka* — koder jednim prolazom može odrediti sve vjerojatnosti, ali ih onda mora poslati u dekoder, zajedno s kodiranim podacima.
- *Dinamičan nad svim nizovima poruka* — koder ažurira svoje vjerojatnosti kako kodira poruke. Nije nužno slati vjerojatnosti dekoderu, jer i dekoder može ažurirati svoje vjerojatnosti prema primljenim porukama, ali tek nakon dekodiranja. Zato je bitno da koder kodira poruku sa starim vjerojatnostima, a zatim ažurira vjerojatnosti prema toj poruci. U protivnom, dekoder ne bi imao točne vjerojatnosti pri dekodiranju te poruke.

3.1 Kodiranje duljine javljanja

Kodiranje duljine javljanja (eng. *Run-length Coding*) je najjednostavnija shema kodiranja koja uzima u obzir kontekst. Osnovna ideja je zamijeniti podnizove istih poruka u nizu, nizom uređenih parova, koje čine vrijednost poruke i broj koliko se puta za redom ta vrijednost ponavlja na tom mjestu u originalnom nizu. Na primjer, poruka `acccbbaaabb` transformirala bi se u `(a,1), (c,3), (b,2), (a,3), (b,2)`. Nakon transformacije, može se koristiti vjerojatnosni koder (primjerice, *Huffmanov koder*) kako bi se kodirale vrijednosti poruka i brojevi ponavljanja.

3.2 Kodiranje pomakom na početak

Kodiranje pomakom na početak (eng. *Move-To-Front Coding*) transformira niz poruka u niz prirodnih brojeva, koji se onda kodira Huffmanovim ili aritmetičkim koderom. U praksi su ta dva koraka isprepletena, ali mi ćemo sagledati svaki korak posebno. Pretpostavljamo da sve poruke dolaze iz iste abecede. Niz poruka prebacujemo u niz prirodnih brojeva prolazeći kroz niz i za svaku poruku vraćamo njezinu poziciju u abecedi, a zatim tu poruku prebacujemo na početak abecede.

Na primjer, kodiranje poruke `b`, s trenutnim poretkom abecede `[a,b,c,...]`, vratilo bi broj `2`, a poredak abecede bi se promijenio u `[b,a,c,...]`. Na isti način, za niz `b,b,c,b` dobivamo kod `2,1,3,2`. Dobiveni niz prirodnih brojeva se tada može kodirati Huffmanovim ili aritmetičkim koderom. Ideja je da će se isti znakovi često pojavljivati jedni blizu drugih u poruci, pa će se mali prirodni brojevi često pojavljivati. To će dovesti do nejednolike vjerojatnosne distribucije brojeva i rezultirati boljom kompresijom.

3.3 Predviđanje djelomičnim podudaranjem

Ideja *predviđanja djelomičnim podudaranjem* (eng. *Prediction by Partial Matching, PPM*) je iskoristiti poznavanje prethodnih k znakova kako bi se generirala uvjetna vjerojatnost trenutnog znaka. Najjednostavnija verzija je da za svaki mogući niz s duljine k znakova, čuvamo rječnik koji sadrži informacije koliko je puta svaki znak x slijedio nakon s. Uvjetna vjerojatnost od x u kontekstu s je tada $C(x|s)/C(s)$, gdje je $C(x|s)$ broj koliko puta je x bio nakon s, a $C(s)$ je broj koliko puta se s pojavio. Nakon generiranja vjerojatnosne distribucije na opisani način, Huffmanovim ili aritmetičkim koderom kodiramo niz.

Nije nužno slati podatke o distribuciji zajedno s kodom, jer kontekst prethodi znaku koji se dekodira pa je on već dekodiran i poznat. Na temelju tog konteksta, jednako kao i koder, dekoder može odrediti vjerojatnosnu distribuciju koju treba koristiti. Kod ovog pristupa, vjerojatnosti su često visoke pa je puno bolje koristiti *aritmetički koder*.

Problemi s opisanom jednostavnom verzijom su:

1. Rječnici mogu postati jako veliki.
2. Slučaj kada je broj pojavljivanja znaka x nakon s jednak 0.

Problem velikih rječnika ne možemo rješiti, ali ga možemo umanjiti tako da biramo male k. Kad je broj pojavljivanja znaka x nakon s jednak 0, ne možemo vjerojatnost postaviti na 0, iako je $C(x|s) = 0$, jer bismo onda tom znaku trebali pridijeliti kod beskonačne duljine.

Predviđanje djelomičnim podudaranjem gradi rječnik, počevši s praznim rječnikom. Svaki put kada algoritam dođe do niza znakova kojeg još nije vidio, pokuša naći niz znakova koji je za jedan kraći. Ako ni njega ne nađe, dalje skraćuje kontekst, dok ne dođe do

duljina 0		*duljina 1*		*duljina 2*	
Kontekst	**Ponavljanja**	**Kontekst**	**Ponavljanja**	**Kontekst**	**Ponavljanja**
prazan	a = 4	a	c = 3	ac	b = 1
	b = 2	b	a = 2		c = 2
	c = 5	c	a = 1	ba	c = 1
			b = 2	ca	a = 1
			c = 2	cb	a = 2
				cc	a = 1
					b = 1

Tablica 3.1: Primjer podataka koje generira predviđanje djelomičnim podudaranjem za $k = 2$ i niz znakova accbaccacba.

podudaranja. U slučaju iz drugog problema, gleda se broj pojavljivanja tog znaka bez da se gleda kontekst. Za svaku duljinu iz $\{0, 1, \ldots, k\}$ čuvaju se svi konteksti te duljine koji su se pojavili, a za svaki kontekst pamti se koliko puta se koji znak pojavio nakon njega.

Primjer podataka koji se čuvaju za $k = 2$ i niz znakova accbaccacba možete vidjeti u tablici 3.1. Kontekst nakon tog niza znakova je ba.

Pogledajmo gdje će se pronaći podudaranja za razne znakove koji mogu slijediti nakon niza znakova accbaccacba. Uzmimo da sljedeći znak dolazi iz abecede {a,b,c,d}.

- a: nakon ba nikad nije slijedio a
  ↪ nakon a nikad nije slijedio a
  ↪ a se pojavio 4 puta ako ne gledamo kontekst.

- b: nakon ba nikad nije slijedio b
  ↪ nakon a nikad nije slijedio b
  ↪ a se pojavio 2 puta ako ne gledamo kontekst.

- c: nakon ba je jednom slijedio c.

- d: nakon ba nikad nije slijedio d
  ↪ nakon a nikad nije slijedio d
  ↪ d se nikad nije pojavio
  ↪ d se dodjeljuje vjerojantost tako da svi znakovi koji se još nisu pojavili imaju istu vjerojatnost.

Dekoder ne zna za koju duljinu konteksta treba gledati kontekst, jer mu nije poznat znak koji dekodira pa ne zna kada treba gledati kraći kontekst. Kako bi mu prenio tu informaciju, koder, prije slanja kodiranog znaka, za svaki put kada je on pri traženju podudaranja prešao u kraći kontekst, šalje po jedan poseban znak koji označava prelazak u kraći kontekst.

Kada prelazi u kraći kontekst, algoritam predviđanja djelomičnim podudaranjem može iskoristiti činjenicu da je prešao u kraći kontekst, kako bi isključio neke znakove iz kraćeg konteksta i time povećao vjerojatnosti ostalih znakova. U konačnici, to smanjuje duljinu koda.

Na primjer, ako nakon niza znakova iz tablice 3.1 slijedi znak `a`, bit će poslan znak za prelazak u kraći kontekst. Na temelju te činjenice, znamo da znak `c` nije znak koji slijedi, jer, u suprotnom, ne bismo morali slati znak za prelazak u kraći kontekst, nego bismo već za duljinu 2 pronašli podudaranje.

Poglavlje 4

Lempel-Ziv algoritmi

Lempel-Ziv algoritmi tijekom kompresije grade rječnik dosad viđenih nizova znakova i koriste te informacije kako bi kodirali zadani niz poruka. Mjesto u nizu znakova na kojem se algoritam nalazi nazivat ćemo *pokazivač*. Algoritam gleda koji je najdulji niz znakova koji počinje na pokazivaču, a već se nalazi u rječniku. Tada vraća neki kod koji jednoznačno označuje to podudaranje i nastavlja s kodiranjem nakon tog niza.

Ziv i Lempel su 1977. i 1978. godine, u odvojenim člancima, opisali algoritme koji su po autorima i godinama kada su članci objavljeni nazvani *Lempel-Ziv 77* (LZ77) i *Lempel-Ziv 78* (LZ78). Postoji više optimizacija LZ77 i LZ78, a neke od poznatih su LZSS (Storer i Szymanski), LZFG (Fiala i Greene), LZRW (Williams) i LZW (Welch).

4.1 Lempel-Ziv 77

LZ77 koristi klizni prozor, podijeljen u dva dijela — dio prije pokazivača koji zovemo *rječnik* i dio koji počinje na pokazivaču. Klizni prozor se pomiče zajedno s pokazivačem. Duljine dijelova prozora su fiksne za vrijeme izvršavanja algoritma i obično se radi o relativno velikim brojevima, kako bi se mogla pronaći što veća podudaranja.

Algoritam ima tri koraka koji se ponavljaju u petlji:

1. Nađi najdulji niz znakova koji počinje na pokazivaču i u potpunosti je sadržan u dijelu prozora koji počinje na pokazivaču, a nalazi se i u rječniku.

2. Vrati trojku (p, n, c), gdje je p pozicija podudaranja u rječniku (brojimo unazad, počevši od pokazivača), n duljina podudaranja, a c znak koji slijedi nakon niza znakova kojem smo pronašli podudaranje.

3. Pomakni pokazivač za $n + 1$ mjesta prema naprijed.

Korak	Ulazni niz	Izlazni kod
1.	**a** a c a a c a b c a b a a a c	(0, 0, a)
2.	a **a** c a a c a b c a b a a a c	(1, 1, c)
3.	a a c **a** a c a b c a b a a a c	(3, 4, b)
4.	a a c a a c a b **c** a b a a a c	(3, 3, a)
5.	a a c a a c a b c a b a **a** a c	(1, 2, c)

Tablica 4.1: Primjer rada kodera LZ77 algoritma s prozorom ukupne veličine 10 i rječnikom veličine 6. Prozor ima žutu pozadinu, pokazivač je podebljan, a dio prozora koji nije u rječniku je podcrtan.

Primjer rada algoritma možete vidjeti u tablici 4.1. Napomenimo da u slučajevima kada je $p < n$, koder uzme p znakova prije pokazivača i ponovi ih $\lceil (n-p)/p \rceil$ puta nakon pokazivača, kako bi imao dovoljno podataka da popuni n pozicija. Takav slučaj se dogodio u 3. koraku u tablici 4.1.

Opišimo još rad dekodera. Ako je dekoder dekodirao dio niza do trenutne pozicije, on ima isti rječnik kojeg je na toj poziciji imao i koder. Na temelju trojke (p, n, c) određuje sljedećih $n + 1$ znakova, tako što pogleda niz znakova duljine n koji počinje na poziciji za p mjesta lijevo od pokazivača i vrati taj niz. Nakon toga, vrati još c i pomakne pokazivač za $n + 1$ mjesta udesno. Ovaj korak ponavljamo do kraja niza.

4.2 Lempel-Ziv 78

LZ78 drukčije pristupa problemu predstavljanja prethodnog dijela niza znakova. Umjesto prozora, on gradi rječnik nizova znakova prilikom kodiranja, počevši od praznog rječnika. Samo jedan cijeli broj (indeks) je dovoljan da bi smo specificirali neki niz znakova iz rječnika te ne treba posebno specificirati duljinu tog niza. Zbog toga, izlaz nije niz trojki, nego niz parova. Prva vrijednost u paru je cijeli broj, koji označuje indeks niza znakova u rječniku, a druga je znak koji slijedi nakon tog niza znakova (specificiranog indeksom). Ukoliko u rječniku nije pronađeno podudaranje, prva vrijednost u paru bit će jednaka 0. U rječnik dodajemo novi niz znakova, koji odgovara pronađenom nizu znakova s dodatnim znakom na kraju — onim koji slijedi nakon tog niza (drugi dio para). Drugim riječima, dodani novi niz je onaj kojeg smo kodirali u tom koraku. Kao i u LZ77, pokazivač se nakon vraćanja jednog para, pomiče za $n + 1$ mjesto udesno, gdje je n duljina niza specificiranog vraćenim indeksom, tj. iza kodiranog niza. Dekoder radi na način analogan dekoderu kod LZ77.

Na početku rada rječnik je prazan, a kako u procesu kompresije raste, indeksi postaju sve veći brojevi i zahtijevaju sve više bitova za kodiranje. Razne verzije LZ78 koriste

Koder			Rječnik	
Korak	**Niz za kodirati**	**Izlazni kod**	**Indeks**	**Niz znakova**
1.	aacaacabcabaaac	(0, a)	1	a
2.	acaacabcabaaac	(1, a)	2	aa
3.	aacabcabaaac	(2, c)	3	aac
4.	abcabaaac	(1, b)	4	ab
5.	cabaaac	(0, c)	5	c
6.	abaaac	(4, a)	6	aba
7.	aac	(2, c)		

Tablica 4.2: Primjer kodiranja niza aacaacabcabaaac LZ78 algoritmom.

različite strategije za rješavanje tog problema. One se svode na pražnjenje rječnika kada se zadovolji neki uvjet. Na primjer, pri kompresiji u GIF, rječnik se prazni kada dosegne određenu veličinu, u Unix Compressu kada više nije efikasan, a u British Telecom Standardu, kada dosegne određenu veličinu, prvo se izbacuju oni nizovi znakova koji najdulje nisu bili korišteni.

Primjer rada LZ78 algoritma možete vidjeti u tablici 4.2. Velika prednost LZ78 je što, dobrom implementacijom rječnika, kodiranje i dekodiranje mogu biti linearne složenosti u duljini poruke [6].

Poglavlje 5

Burrows-Wheelerov algoritam

Burrows i Wheeler su 1994. godine predložili algoritam za kompresiju podataka koji se temelji na *Burrows-Wheelerovoj transformaciji*. Omjeri kompresije koje je postizao taj algoritam bili su usporedivi s najboljim poznatim algoritmima. Implementacija Burrows-Wheelerovog algoritma zvana `bzip`, još uvijek je jedan od najboljih algoritama za kompresiju teksta.

5.1 Koder

Koraci originalnog Burrows-Wheelerovog algoritma su sljedeći:

1. Burrows-Wheelerova transformacija ulaznog niza (poruke).
2. Kodiranje pomakom na početak.
3. Kodiranje Huffmanovim ili aritmetičkim koderom.

Valja napomenuti da je Wheeler predložio korištenje kodiranja duljine javljanja za nule, kao zadnji korak prije kodiranja Huffmanovim ili aritmetičkim koderom, jer u pojedinim dokumentima nule čine velik dio podataka, ali taj korak nije bio uključen u objavljeni algoritam.

Neka je x niz znakova duljine n. Burrows-Wheelerova transformacija (BWT) generira n nizova iste duljine n, tako što je i-ti niz znakova dobiven rotacijom niza x za $i-1$ mjesta ulijevo. Time dobivamo matricu znakova $M(x)$. Slijedi primjer matrice M za riječ *matematika*:

$$M(matematika) = \begin{bmatrix} m & a & t & e & m & a & t & i & k & a \\ a & t & e & m & a & t & i & k & a & m \\ t & e & m & a & t & i & k & a & m & a \\ e & m & a & t & i & k & a & m & a & t \\ m & a & t & i & k & a & m & a & t & e \\ a & t & i & k & a & m & a & t & e & m \\ t & i & k & a & m & a & t & e & m & a \\ i & k & a & m & a & t & e & m & a & t \\ k & a & m & a & t & e & m & a & t & i \\ a & m & a & t & e & m & a & t & i & k \end{bmatrix}.$$

Matrica $M(x)$ se tada transformira u matricu $M'(x)$, sortiranjem redova matrice u leksikografskom poretku. Slijedi primjer matrice M' za riječ *matematika*:

$$M'(matematika) = \begin{bmatrix} a & m & a & t & e & m & a & t & i & \mathbf{k} \\ a & t & e & m & a & t & i & k & a & \mathbf{m} \\ a & t & i & k & a & m & a & t & e & \mathbf{m} \\ e & m & a & t & i & k & a & m & a & \mathbf{t} \\ i & k & a & m & a & t & e & m & a & \mathbf{t} \\ k & a & m & a & t & e & m & a & t & \mathbf{i} \\ \underline{m} & \underline{a} & \underline{t} & \underline{e} & \underline{m} & \underline{a} & \underline{t} & \underline{i} & \underline{k} & \underline{\mathbf{a}} \\ m & a & t & i & k & a & m & a & t & \mathbf{e} \\ t & e & m & a & t & i & k & a & m & \mathbf{a} \\ t & i & k & a & m & a & t & e & m & \mathbf{a} \end{bmatrix}.$$

Rezultati Burrows-Wheelerove transformacije su niz znakova x^{bwt}, kojeg dobivamo iščitavanjem zadnjeg stupca matrice $M'(x)$, i redni broj retka u matrici $M'(x)$, čiji su elementi jednaki početnom nizu x. Označavamo ga s $R(x)$. U navedenom primjeru vidimo da je $matematika^{bwt} = kmmttiaeaa$ i $R(matematika) = 7$.

Niz x^{bwt} se kodira kodiranjem duljine javljanja, a nakon toga Huffmanovim ili aritmetičkim algoritmom. Valja napomenuti da se i $R(x)$ sprema, jer je on nužan dekoderu da bi mogao rekonstruirati x iz x^{bwt}.

5.2 Dekoder

Dekoder Burrows-Wheelerovog algoritma radi inverzne radnje radnjama iz kodera i radi ih u obrnutom poretku. Tako su njegovi koraci sljedeći:

1. Dekoder Huffmanovog, odnosno, aritmetičkog algoritma.

2. Dekoder kodiranja pomakom na početak.

3. Inverzna Burrows-Wheelerova transformacija.

Opišimo rad inverzne Burrows-Wheelerove transformacije. Ulazni podaci inverzne BWT su x^{bwt} i $R(x)$. Primijetimo da zadnji znak u x mora biti $R(x)$-ti znak u nizu x^{bwt}. Također, primijetimo da je x^{bwt} permutacija niza x. Ako leksikografski sortiramo x^{bwt}, dobit ćemo niz znakova koji odgovara prvom stupcu matrice $M'(x)$. Dakle, prvi znak niza x odgovara $R(x)$-tom znaku sortiranog niza x^{bwt}. Time smo dekodirali prvi znak u x.

Zatim, u zadnjem stupcu $M'(x)$ (odnosno, u x^{bwt}) tražimo redak u kojem se prvi puta pojavljuje znak koji je jednak prvom znaku u x, tako da preskačemo znakove koji su već iskorišteni, kao što je znak u retku $R(x)$. Kada odredimo redni broj tog retka, označimo ga s i. Tada znamo da je i-ti znak u zadnjem stupcu, ujedno i drugi znak u nizu x. Ovaj postupak ponavljamo dok ne odredimo cijeli niz x. U tablici 5.1 ilustriran je rad dekodera za niz *matematika*.

Sort(matematika)	**matematika^{bwt}**	**Redni broj u nizu**
a	k	9
a	m	1
a	m	5
e	t	3
i	t	7
k	i	8
m	**a**	10
m	e	4
t	a	2
t	a	6

Tablica 5.1: Ilustracija rada dekodera Burrows-Wheelerovog algoritma koji dekodira riječ *matematika*.

Poglavlje 6

Usporedba kompresije slika u PNG i JPEG formatima

PNG

PNG (Portable Network Graphics) je format za spremanje slika koji koristi *lossless* kompresiju i podržava 24-bitnu RGB ili 32-bitnu RGBA paletu boja i transparentnost. Kod spremanja slika koje sadrže tekst ili grafike, odnosno, slika s naglim promjenama u boji i velikim područjima s istom bojom, slike u PNG formatu mogu biti čak i manje nego u JPEG formatu. Budući da PNG koristi lossless kompresiju, slike u PNG formatu ne sadrže vizualne mane u područjima s velikim kontrastom.

Kompresija u PNG ima dvije faze. U prvoj fazi, na podacima se radi filtriranje predikcijskom metodom. U drugoj fazi, PNG za kompresiju podataka koristi *DEFLATE*, nepatentiran lossless algoritam za kompresiju podataka, koji koristi kombinaciju *Lempel-Ziv 77* (LZ77) i *Huffmanovog kodiranja*.

JPEG

JPEG (Joint Photographic Experts Group) je format za spremanje slika koji koristi *lossy* kompresiju i najčešće se koristi za spremanje digitalnih fotografija. JPEG radi na 24-bitnoj RGB paleti boja, ali pri spremanju slika u JPEG format, ne koristi RGB model boja, nego se boje konvertiraju u model Y'C_BC_R. Komponenta Y' predstavlja svjetlinu piksela, a C_B i C_R podjelu u plave i crvene komponente (eng. *chrominance*).

Prvi korak JPEG algoritma je prebacivanje slike iz RGB modela u Y'C_BC_R model. Zatim se slika dijeli u 8×8 blokove, za svaku komponentu modela posebno, kako bi se svaka posebno kodirala. Na svaki blok se primijeni diskretna kosinusna transformacija i onda skalarana kvantizacija, kod koje dolazi do gubitka podataka. Uz prebacivanje u Y'C_BC_R model, to je jedini dio algoritma u kojem dolazi do gubitka podataka. Naime, kod

skalarne kvantizacije, nakon dijeljenja, dolazi do zaokruživanja na najbliži cijeli broj. DC komponenta iz kvantizacije ne kodira se izravno, nego se posebno kodira DC komponenta prvog bloka, a za ostale se kodira razlika od prethodnog. Te razlike se kodiraju koristeći *Huffmanovo ili aritmetičko kodiranje*. Ostale komponente se kodiraju koristeći *kodiranje duljine javljanja*, a zatim *Huffmanovim ili aritmetičkim kodiranjem*.

RAISE-1k

Skup slika *RAISE-1k* je podskup većeg skupa slika *RAISE (Raw Images Dataset)* koji je objavljen u članku [7]. RAISE-1k može se pronaći na [5]. RAISE-1k se sastoji od tisuću slika, koje se mogu preuzeti s interneta u TIFF ili NEF formatu. Za potrebe ovog rada, preuzete su sve slike iz RAISE-1k, koje su bile spremljene u lossless TIFF formatu. Preuzeto je 728 ispravnih slika. Na slici 6.1 prikazane su neke od preuzetih slika, nakon konverzije u JPEG format.

(a) r0fb0a690t.jpg

(b) r01a31693t.jpg

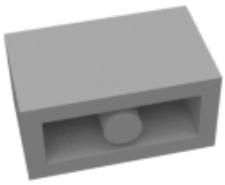
(c) r1c24a571t.jpg

(d) r1dc4077at.jpg

Slika 6.1: Primjeri slika iz RAISE-1k u JPEG formatu (nakon kompresije).

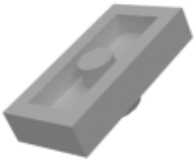
LEGO brick images

Skup slika *LEGO brick images* je skup slika LEGO kocaka generiranih računalom i spremljenih u PNG formatu, a možete ga pronaći na [3]. Preuzeto je 12758 slika.

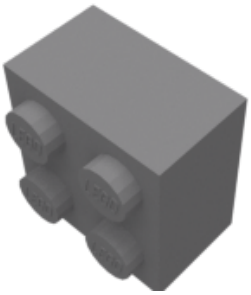


(a) train/3004 Brick 1x2/0212.png

(b) train/3673 Peg 2M/0286.png



(c) train/3794 Plate 1X2 with 1 Knob/ 201706161606-0344.png



(d) valid/3003 Brick 2x2/0199.png

Slika 6.2: Primjeri slika iz LEGO brick images u PNG formatu.

Rezultati

U tablici 6.1 možete vidjeti kako se odnose prosječne veličine slika iz RAISE-1k i LEGO brick images skupova slika, u različitim formatima. Sve veličine su u bajtovima.

Rezultati iz tablice dobiveni su Console aplikacijom u jeziku C#, koristeći metode iz `System.Drawing.Image`. Za RAISE-1k, na temelju slika u TIFF formatu, stvorene su slike u PNG i JPEG formatu. Za LEGO brick images, iz PNG formata stvorene su slike samo u JPEG formatu. Očitala bi se veličina izvorne slike i tek stvorenih slika, koristeći `Length` svojstvo objekta klase `System.IO.FileInfo`. Potom bi se generirane slike obrisale, kako ne bi zauzimale prostor na disku računala.

	TIFF	**PNG**	**JPEG**
RAISE-1k	23541481.77	32375886.88	1883507.23
LEGO brick images	-	14533.51	2904.54

Tablica 6.1: Usporedba prosječne veličine slika iz RAISE-1k i LEGO brick images skupova slika, u TIFF, PNG i JPEG formatima (u bajtovima, zaokruženo na dvije decimale).

Prosječna veličina slike iz RAISE-1k u JPEG formatu odgovara 5.82% veličine iste slike u PNG formatu (zaokruženo na dvije decimale), dok je taj postotak kod LEGO brick images 19.99%. Drugim riječima, slike iz RAISE-1k su približno 17 puta veće u PNG formatu, nego u JPEG formatu, dok su slike iz LEGO brick images približno 5 puta veće u PNG formatu, nego u JPEG formatu.

Iz dobivenih rezultata za oba skupa slika, primjećujemo da su slike u JPEG formatu manje no što je očekivano, s obzirom na to da PNG koristi lossless kompresiju, a JPEG lossy kompresiju.

Također, uočavamo da postoje znatne razlike u omjerima veličina slika u PNG i JPEG formatima među skupovima slika. Tu razliku možemo pripisati činjenici da slike u skupu RAISE-1k sadrže puno više detalja i različitih boja pa ih je moguće znatno smanjiti, bez da se znatno utječe na vizualnu kvalitetu slike.

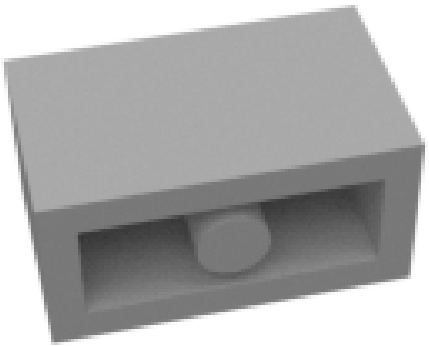
Usporedimo sada vizualnu kvalitetu slika u različitim formatima. Na slici 6.3 možemo vidjeti sliku imena ”r0fb0a690t” iz RAISE-1k, u PNG i JPEG formatima. Na nekim mjestima na slici može se primijetiti razlika u bojama. Na primjer, boja debla desnog drveta i hlača čovjeka sa šeširom koji pleše, ali to su manje razlike, koje je teško uočiti. Ova slika u JPEG formatu je 16.68 puta manja od one u PNG formatu.

Na slici 6.4 možemo vidjeti sliku ”train/3004 Brick 1x2/0212” iz LEGO brick images, u PNG i JPEG formatima. Vizualne razlike između slika sada su lako uočljive. Pozadina slike u PNG formatu je bijela, a ona u JPEG formatu je crna. Razlog za to je što je pozadina slike u PNG formatu zapravo prozirna, a JPEG ne podržava prozirnost pa je dio slike, koji bi trebao biti proziran, ovdje crn. Također, vidimo kako rub objekta na slici u JPEG formatu nije gladak, kao što bi trebao biti. Kvaliteta slike u JPEG formatu teško se može smatrati zadovoljavajućom, a ona je samo 4.83 puta manja od iste slike u PNG formatu.

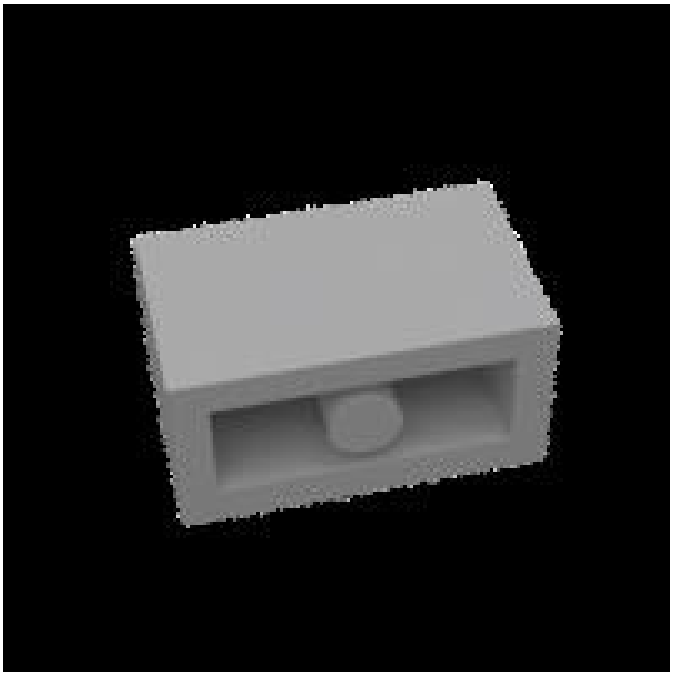
(a) r0fb0a690t.png

(b) r0fb0a690t.jpg

Slika 6.3: Usporedbe iste slike iz RAISE-1k, u PNG i JPEG formatima.



(a) train/3004 Brick 1x2/0212.png



(b) train/3004 Brick 1x2/0212.jpg

Slika 6.4: Usporedbe iste slike iz LEGO brick images, u PNG i JPEG formatima.

Bibliografija

[1] *Entropy (information theory)*, https://en.wikipedia.org/wiki/Entropy_(information_theory), posjećena srpanj 2019.

[2] *JPEG*, https://en.wikipedia.org/wiki/JPEG, posjećena srpanj 2019.

[3] *LEGO brick images*, https://www.kaggle.com/joosthazelzet/lego-brick-images, posjećena srpanj 2019.

[4] *Portable Network Graphics*, https://en.wikipedia.org/wiki/Portable_Network_Graphics, posjećena srpanj 2019.

[5] *RAISE-1k*, http://loki.disi.unitn.it/RAISE/confirm.php?package=1k, posjećena srpanj 2019.

[6] G. E. Blelloch, *Introduction to Data Compression*, Computer Science Department, Carnagie Mellon University, 2013, https://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/compression.pdf.

[7] D. T. Dang-Nguyen, C. Pasquini, V. Conotter i G. Boato, *RAISE - A Raw Images Dataset for Digital Image Forensics*, ACM Multimedia Systems, Portland, Oregon (18.-20.3.2015.).

[8] S. Deorowicz, *Universal lossless data compression algorithms*, PhD thesis, Silesian University of Technology, Gliwice, Poland, 2003, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.128.6840&rep=rep1&type=pdf.

Sažetak

Kompresija podataka je proces koji smanjuje veličinu podataka, tako da uklanja suvišne dijelove. Svaki zadatak kompresije podataka sastoji se od dva dijela, algoritma za kompresiju podataka i algoritma za dekompresiju podataka. U ovom radu dane su Shannonove definicije entropije i vlastite informacije poruke. One predstavljaju ključne pojmove teorije informacija, u sklopu koje su razvijeni algoritmi za kompresiju podataka.

Opisano je prefiksno kodiranje te Huffmanov koder, kao najvažniji prefiksni koder. Opisan je i aritmetički koder. Oba kodera su često korištena u mnogim algoritmima za kompresiju podataka te opisujemo njihovu ulogu u kodiranju duljine javljanja, kodiranju pomakom na početak i predviđanju djelomičnim podudaranjem, kao i pripadne algoritme.

Posljednje dvije skupine algoritama za kompresiju koje razmatramo u ovom radu su Lempel-Ziv algoritmi i Burrows-Wheelerov algoritam. Lempel-Ziv algoritmi, tijekom kompresije, grade rječnik dosad viđenih nizova znakova i koriste te informacije kako bi kodirali zadani niz poruka. Burrows-Wheelerov algoritam transformira ulazne podatke Burrows-Wheelerovom transformacijom te na transformiranim podacima primjenjuje kodiranje pomakom na početak, a zatim Huffmanov ili aritmetički koder.

Za kraj, uspoređujemo dva konkretna formata za spremanje slika, PNG i JPEG format. Usporedbom slika u oba formata na slikama iz Raise-1k skupa slika slikanih kamerom te računalno generiranim slikama iz Lego brick images skupa slika, nameće se zaključak kako je JPEG uistinu bolji format za spremanje slika slikanih kamerom, jer kao lossy algoritam može postići daleko bolje omjere kompresije, a razlika u kvaliteti slike u odnosu na lossless PNG format je mala. S druge strane, za računalno generirane slike, kao što su slike iz Lego brick images, PNG format je očito bolji, jer kod JPEG formata dolazi do jasno vidljivog pada u kvaliteti slika, a razlika u omjerima kompresije nije velika, kao kod slika slikanih kamerom.

Summary

Data compression is a process that reduces the size of the data by removing redundant parts. Each data compression task consists of two parts, an encoding algorithm and a decoding algorithm. This thesis provides Shannon's definitions of entropy and self-information of a message. They represent the key concepts of information theory within which data compression algorithms have been developed.

Prefix coding is described and so is the Huffman Coder, as the most important prefix coder. An Arithmetic Coder is also described. Both encoders are often used in many data compression algorithms, and we describe their role in Run-Length Coding, Move-To-Front Coding, and Partial Match Prediction, as well as the algorithms themselves.

The last two sets of compression algorithms we consider in this thesis are the Lempel-Ziv algorithms and the Burrows-Wheeler algorithm. During compression, the Lempel-Ziv algorithms build a dictionary of character strings seen so far, and use this information to encode a given string of messages. The Burrows-Wheeler algorithm transforms input data by using the Burrows-Wheeler transform, and applies the Huffman or arithmetic encoder to the transformed data.

In the end, we compare two specific image saving formats, PNG and JPEG. By comparing images in both formats on images from the Raise-1k dataset created with a digital camera, and computer-generated images from the Lego brick images dataset, we conclude that JPEG is indeed a better format for storing camera images, as it can achieve far better compression ratios, because it is a lossy algorithm, and the difference in image quality compared to the lossless PNG format is small. On the other hand, for computer-generated images, such as images from the Lego brick dataset, the PNG format is clearly better, because JPEG format results in a visibly noticeable drop in image quality, and the difference in compression ratios is not as large as for the camera images.

Životopis

Rođen sam 30. 5. 1995. godine u Zagrebu, Republici Hrvatskoj.

Školovanje sam započeo 2002. godine u osnovnoj školi "Ivan Mažuranić" u Zagrebu. Kroz sve razrede osnovne škole bio sam odličan učenik.

2010. godine upisao sam prirodoslovno-matematički smjer III. Gimanzije u Zagrebu. Tijekom cijelog srednjoškolskog obrazovanja održao sam odličan uspjeh, a na maturi iz matematike postigao sam stopostotni rezultat.

Studij Matematike na Matematičkom odsjeku Prirodoslovno-Matematičkog fakulteta u Zagrebu upisao sam 2014. godine. Tijekom druge i treće godine studija bio sam demonstrator iz kolegija Matematička analiza 1 i Matematička analiza 2.

Preddiplomski studij matematike završio sam 2017. godine. Iste godine upisao sam diplomski studij Računarstvo i matematika.