kompression

Johannes Jørgensen
S2o

Maj 2022

Contents

1 introduktion

Denne PDF-fil du læser i, har en filstørrelse på omkring 1.2 KB. Et gennemsnitligt billede med billedformaten JPG har en størrelse på 11.8 KB,[5] og en time lang Fuld HD-video har en størrelse på 1.2 GB.[6] Det er mindre effektivt, dyre og langsommere at sende større filer verden rundt via internettet. Men det er her kompression kommer ind i billedet. Kompression er en måde at reducere data, at indkode færre bits end den originale repræsentation af givende data. Hver kompression af et stykke data kan enten være "Lossy" eller "Lossless".[7] Lossless kompression er en reduktion af data uden noget data går tabt, hvor lossy kompression er hvor noget af den originale data bliver slette.

2 LZ77 Algoritme

LZ77 (også kaldt LZ1) algoritme er en lossless data kompression algoritme udviklet af Lempel og Ziv i 1977. Der er flere forskellige variationer af LZ77 algoritmen (LZ78 samt LZSS) og algoritmerne kan bruges til forskellige formål. Men alle af LZ-algoritmerne grundlæggende princip er det samme. Algoritme-serien er adopteret af mange forskellige systemer, mest kendt i GIF og DEFLATE algoritmen som bruges i ZIP og PNG.[9]

Den grundlæggende algoritme LZ77 komprimerer gentagende data med at referater, som tidligere eksisterede data i den ukomprimerede data. Referatet til det gentaget data bliver til et talpar, dette kan også kaldes for en "pointer". Udsagnet "pointer" er et tal (en integrer) som refererer til et stykke data's adresse i hukommelsen. Talparet er kaldt "length-distance pair", som svare til udsagte "hvert af de næste længdetegn er lig med tegnene nøjagtigt afstandstegn bagved i den ukomprimerede strøm". Talparet er et output af tal ((D)istance, (L)ength, (c)harater) som er defineret ved:

$$[\boldsymbol{D}, \boldsymbol{L}, \boldsymbol{c}]$$

- (D): Afstanden for antal positioner, der flyttes bagud for at finde starten af den matchende data.
- (L): Længden af det matchende data.
- (c): Karakteren som er repræsenteret efter det matchende data er fundet.

LZ77 algoritmen itererer (køre igennem/gentag) igennem et stykke data, for at finde den længste sekvens af data som bliver brugt flere gange. Sker der et match i et par datastykker, bliver de matchene datastykker gemt.

2.1 Eksempel og gennemgang

En måde at kunne visualisere LZ77 algoritmen er med en gennemgang af et eksempel. For at kunne simplificere hvad algoritmen gør, bruger jeg en streng med gentagende bogstaver. LZ77 er ikke kun brugbar i tekst kompression, men også andre dataformater.[4]
Eksempel streng er til kompression:

$$\boldsymbol{AABCBBABC}$$

Når LZ-algoritmen interager igennem strengen, er algoritmens intuition kun til at komprimere allerede fundet bogstaver, og point tilbage til bogstavets placering i den originale streng. Følgende skema viser gennemgangen af komprimeringen af strengen. Hvis algoritmen ikke har set eller fundet data før, bliver outputtet repræsenteret til en $[\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{c}]$ pointer som pointere

til sig selv. Ellers bliver den outputtet til en pointer med "length-distance pair" som nævn før $[\boldsymbol{D}, \boldsymbol{L}, \boldsymbol{c}]$.

Position	Match	Byte	Output
1	⊘	$\boldsymbol{A}$	$[\mathbf{0}, \mathbf{0}, \boldsymbol{A}]$
2	$\boldsymbol{A}$	⊘	$[\mathbf{1}, \mathbf{1}]$
3	⊘	$\boldsymbol{B}$	$[\mathbf{0}, \mathbf{0}, \boldsymbol{B}]$
4	⊘	$\boldsymbol{C}$	$[\mathbf{0}, \mathbf{0}, \boldsymbol{C}]$
5	$\boldsymbol{B}$	⊘	$[\mathbf{2}, \mathbf{1}]$
6	$\boldsymbol{B}$	⊘	$[\mathbf{1}, \mathbf{1}]$
7	$\boldsymbol{ABC}$	⊘	$[\mathbf{5}, \mathbf{3}]$

Det resulterende komprimeret output er:

$$[\mathbf{0}, \mathbf{0}, \boldsymbol{A}][\mathbf{1}, \mathbf{1}][\mathbf{0}, \mathbf{0}, \boldsymbol{B}][\mathbf{0}, \mathbf{0}, \boldsymbol{C}][\mathbf{2}, \mathbf{1}][\mathbf{1}, \mathbf{1}][\mathbf{5}, \mathbf{3}]$$

Hvis vi simplificerer outputtet til at det kun er direkte referater til forrige data, ser det sådan ud:

$$\boldsymbol{A}[\mathbf{1}, \mathbf{1}]\boldsymbol{BC}[\mathbf{2}, \mathbf{1}][\mathbf{1}, \mathbf{1}][\mathbf{5}, \mathbf{3}]$$

For at finde frem til den originale data igen, gøres det samme som ved kompression bare baglæns. Så skidt 1 finder vi et $\boldsymbol{A}$ hvor vi efter får en pointer $[\mathbf{1}, \mathbf{1}]$ i skridt 2, til at gå et skidt tilbage i strengen, som er den tilsvarende byte $\boldsymbol{A}$. Dermed bliver det nye output til $\boldsymbol{AA}$ og så ledes. Følgende tabel viser skridtene til at få det originale data af den kom komprimeret data: Eksemplet ser det ikke ud som om at vores originale data er blevet mere komprimeret, da der

skridt	Pointer	Tilsvarende Byte(s)	Output streng
1	$[\mathbf{0}, \mathbf{0}, \boldsymbol{A}]$	$\boldsymbol{A}$	$\boldsymbol{A}$
2	$[\mathbf{1}, \mathbf{1}]$	$\boldsymbol{A}$	$\boldsymbol{AA}$
3	$[\mathbf{0}, \mathbf{0}, \boldsymbol{B}]$	$\boldsymbol{B}$	$\boldsymbol{AAB}$
4	$[\mathbf{0}, \mathbf{0}, \boldsymbol{C}]$	$\boldsymbol{C}$	$\boldsymbol{AABC}$
5	$[\mathbf{2}, \mathbf{1}]$	$\boldsymbol{B}$	$\boldsymbol{AABCB}$
6	$[\mathbf{1}, \mathbf{1}]$	$\boldsymbol{B}$	$\boldsymbol{AABCBB}$
7	$[\mathbf{5}, \mathbf{3}]$	$\boldsymbol{ABC}$	$\boldsymbol{AABCBBABC}$

er nu kommet flere bytes end før. Hvis et bogstav er 1 byte og en pointer består af 2 tal som er 2 bytes i alt:

$$\boldsymbol{AABCBBABC} = \boldsymbol{A}[\mathbf{1}, \mathbf{1}]\boldsymbol{BC}[\mathbf{2}, \mathbf{1}][\mathbf{1}, \mathbf{1}][\mathbf{5}, \mathbf{3}]$$

$$\Longleftrightarrow$$

$$\mathbf{9}\boldsymbol{bytes} < \mathbf{11}\boldsymbol{bytes}$$

Dog i en større skala bliver dette bedre mening end eksemplet. Hvis vores eksempel var en længere tekst, kan fordelene se bedre ud:

Originale:	*Det nye computerspil er meget optimeret til computeren*	47 bytes
komprimeret:	*Det nye computerspil er meget optimeret til* $[\mathbf{31}, \mathbf{8}]$*en*	41 bytes

3 Huffman coding

Huffman coding er en algorithme for lossless kompression, udviklet af David A, Huffman. Metoden blev først set af offeligheden i 1952 i Huffmans rapport "A Method for the Construction of Minimum-Redundancy Codes".[8]

Den generelle ide bag metoden er at opdele de mest hyppige værdier i et stykke data. Værdierne får uddelt deres egen præfikskode (deres eget symbol eller værdi) i et binary tree. Hyppigheden af forekomsten af værdien, bliver binary tree'et kortere. Dette binary tree bliver også kaldt for et "Huffman tree"[3]

Huffmans tree er bygget op så ledes at hver node (datapunkter i et binary tree eller anden datastruktur), indeholder symbolet selv og vægten (hyppigheden af nodes fremkomst). Noderne er også tilkoblet til to underordnede noder, samt en tilkobling til en overordnet node. Tilkoblingen til noden repræsenterer en tilsvarende bit, tilkoblingen til venstre underordnede node repræsenterer bit *'0'*, hvor højre underordnede node er repræsentere bit *'1'*. Ud fra mængden af nodes i Huffmans tree, er hver tilkobling til hver node udstyret med en sandsynlighed for det symbol de repræsenterer. Symbolerne i Huffmans tree er sorteret efter sandsynlighed, den korteste vej i Huffmans tree er til det hyppigste symbol.

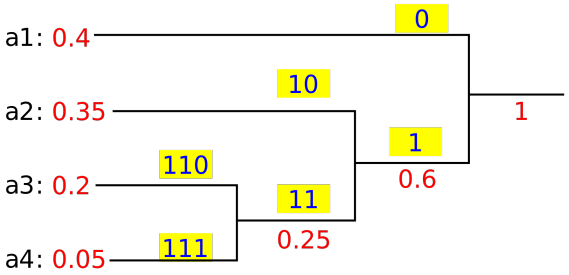

Figure 1: Eksempel på Huffmans Tree på data symbolerne a1, a2, a3, a4 [8]

Figur 1 er et eksempel på Huffmans tree på data symbolerne a1, a2, a3, a4. De røde tal er sandsynligheden for hyppigheden for symbolerne. Hver route til hver node har et repræsenteret bitværdi (*0* eller *1*).
Den repræsenterende kode for symbolerne i Huffmans tree vil blive inddelt ud fra sandsynligheden for hyppigheden for symbolet:

Symbol	**Kode**
a1	0
a2	10
a3	110
a4	111

3.1 Eksempel

Hvis vi tag samme eksempel som brugt i LZ77 algorithmen, ser Huffmans tree således ud:

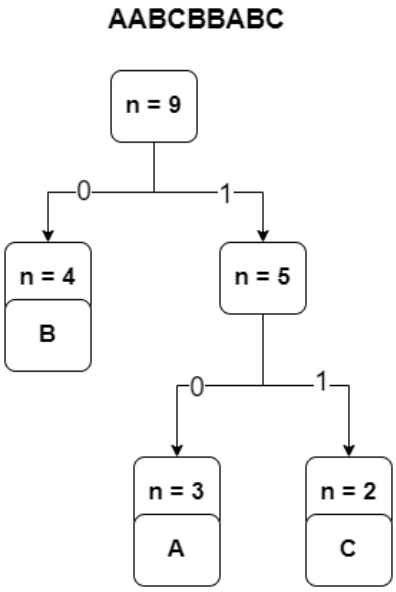

Figure 2: n er mængden af symbolets hyppighed

Den repræsenterende kode for symbolerne i Huffmans tree for eksemplet:

Symbol	Kode
B	0
A	10
C	11

4 Metode og analyse

Der er adskillige kompressionsalgoritmer og metoder. Den mest kendte algoritme inden for komprimering af filer og billeder er DEFLATE algoritmen. DEFLATE algoritmen er en kombination af Huffman coding og LZ77 algoritmerne. Dette opnår fleksibel komprimerings evner som er brugt i forskellige applikationer, mest kendt for deres brug i gzip komprimeret filer og PNG-billedfiler.

4.1 Gennemgang af min kode

Projektet som jeg har lavet, er en virtuel fremvisning af komprimeringen af billeder.[2] Projektet er en hjemmeside som gør det muligt for brugeren at uploade et billede og få det komprimeret. indstillingerne for komprimeringen er både højde, brede og kvalitet af billedet.

Jeg har med brug af et importeret bibliotek "Compressorjs"[1] som bruger browserens indbygget API canvas.toBlob, til kompression. Dette gør at alle billedfiler som bliver uploadet, er der kun mulighed for en lossy komprimering. Til udvikling af hjemmesiden bruger jeg React og TailwindCSS som framework.

Når brugeren uploader en billedfil til hjemmesiden, bliver filen tildelt kompressionsbiblioteket. Bibliotek komprimerer derefter billedfilen ud fra indstillerингerne som brugeren har valgt:

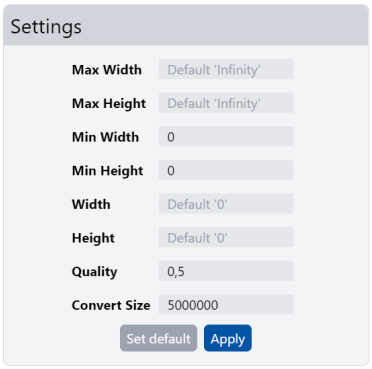

Figure 3: Illustration af 'Settings'

```
if (file.length !== 0) {
  new Compressor(file[0], {
    ...options, // parse user options
    success: (result: File) => {
      // set React hook to result
      setCompressed(result);
    },
    error(err) {
      // error handeling
      console.log(err.message);
    },
  });
}

```

Listing 1: Indsættelse af fil i Compressorjs bibliotek

Det komprimeret billede bliver derefter fremvist til brugeren om både differencen for den komprimeret filstørrelse forhold til det originale billede. Samt får brugeren mulighed til at downloade det komprimeret billede:

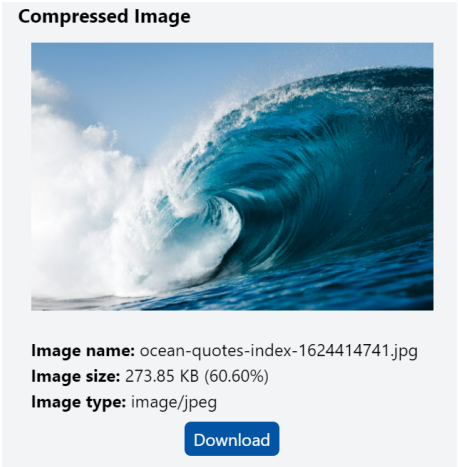

Figure 4: Illustration af 'preview'

```
onClick={() => {
  // get URL from compressed file
  const URL = getImgURL(file);

  fetch(URL).then((res) => {
    const a = document.createElement("a");
    a.style.display = "none";
    a.href = res.url;
    a.download = `${name}`;
    document.body.appendChild(a);
    a.click();
    // Memory cleanup
    window.URL.revokeObjectURL(res.url);
  });
}}

```

Listing 2: Download komprimeret billede med knap

Efter en kompression af et billed for *0,1* af kvaliteten af det originale billed, fås en reduktion på 87% eller 13% af det originale billed. Dette er en kompression fra **451** MB til **58** MB

(a) Det originale billed - før kompression

(b) Det komprimere billed - efter kompression

5 Bilag

```
while input is not empty do
  match := longest repeated occurrence of input that begins in window

  if match exists then
      d := distance to start of match
      l := length of match
      c := char following match in input
  else
      d := 0
      l := 0
      c := first char of input
  end if

  output (d, l, c)

  discard l + 1 chars from front of window
  s := pop l + 1 chars from front of input
  append s to back of window
repeat
```

Listing 3: LZ77 Pseudocode

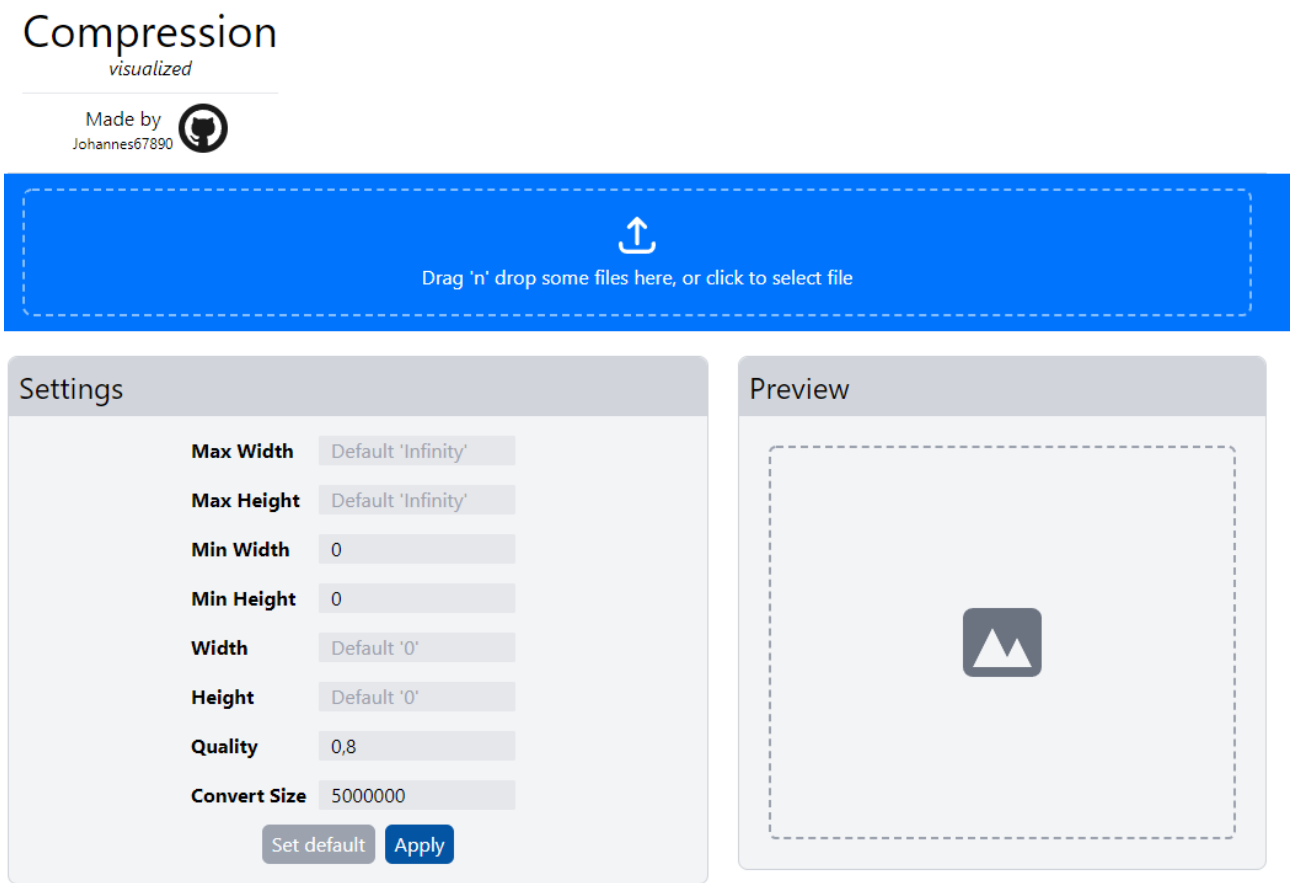

Figure 5: Illustration af Projekte app

Litteraturliste

[1] fengyuanchen. *Compressorjs*. URL: https://github.com/fengyuanchen/compressorjs (visited on 05/22/2020).

[2] Johannes Jørgensen. *VirtualCompressor*. URL: https://github.com/johannes67890/VirtualCompressor (visited on 05/22/2020).

[3] MrBrownCS. *Huffman Coding (Lossless Compression Algorithm)*. URL: https://www.youtube.com/watch?v=NjhJJYHpYsg&ab_channel=MrBrownCS (visited on 05/21/2020).

[4] SoftwareEngenius. *Learn in 5 Minutes: LZ77 Compression Algorithms*. URL: https://www.youtube.com/watch?v=jVcTrBjI-eE&ab_channel=SoftwareEngenius (visited on 05/21/2020).

[5] Solarwinds. *New facts and figures about image format use on websites*. December 2008. URL: https://www.pingdom.com/blog/new-facts-and-figures-about-image-format-use-on-websites/ (visited on 05/14/2020).

[6] John Tkaczewski. *How Big Are Movie Files in 2022?* Januar 2020. URL: https://www.filecatalyst.com/blog/how-big-are-movie-files/ (visited on 05/14/2020).

[7] Wikipedia. *Data compression*. URL: https://en.wikipedia.org/wiki/Data_compression (visited on 05/14/2020).

[8] Wikipedia. *Huffman coding*. URL: https://en.wikipedia.org/wiki/Huffman_coding (visited on 05/21/2020).

[9] Wikipedia. *LZ77 and LZ78*. URL: https://en.wikipedia.org/wiki/LZ77_and_LZ78 (visited on 05/16/2020).