A picture containing table

Description automatically generated

Foreleser: Helge Hafting

Nicolai Thorer Sivesind

Erlend Rønning

Aleksander Brekke Røed

**Komprimering**

**Oppgave 8 – IDATT2101**

08. november, 2021

Innlevering av:

Lempel-Ziv

Teori

Lempel-ziv bruker referanser til sekvenser den kjenner igjen fra tidligere til å komprimere filer. Hvis algoritmen oppdager at en sekvens med tre eller fler bytes repeterer seg fra tidligere i teksten den har analysert, kan den spare plass ved å bruke to bytes for å referere til denne sekvensen fremfor å lagre bytesene som de er. Den første byten for å fortelle hvor den refererte sekvensen starter og den andre for hvor lang den er. Vi kan øke dette til å bruke to bytes for avstand og en byte for lengde. Da dobler vi avstanden algoritmen kan se bakover for å sjekke etter repetert sekvens. Vi ønsker kun å referere til denne hvis sekvensen er på 3 bytes eller mer. Dette er fordi at vi bruker 3 bytes for å lagre referansen uansett, så en referanse til en sekvens på mindre enn dette vil føre til å at vi bruker mer plass enn hva den opprinnelig filen gjorde.

Tester av algoritmen

Ford disse filene komprimerer algoritmen vi har skrevet en god del. Under er skjermdumper fra orginalfil, komprimert og utpakket fil fra komprimert. Alle de utpakkede filene har vi sjekket er lik med orginalfilen ved bruk av UNIX sin terminalkommando «diff»

.txt og .lyx

**Graphical user interface, text

Description automatically generated**

Her ser vi at den komprimerte filen er ca. 5KB mindre enn den orginale filen.

**Graphical user interface, text

Description automatically generated**

Her ser vi at den komprimerte er ca. 79KB mindre enn orginalen.

.pdf

Graphical user interface, text

Description automatically generated

Her ser vi at den komprimerte filen faktisk er litt større enn originalfilen. Dette er fordi PDF-filer allerede er komprimert, så å komprimere disse kan faktisk medføre at den komprimerte blir litt større. Grunnen til det er at hvis man får mange tilfeller der man har en sekvens med ukomprimerte bytes som har en lengde på 1, så bruker man altså 2 bytes for å representere 1 byte. En byte for å indikere at det kommer en sekvens med ukomprimerte bytes, og en for selve byten. Hvis den jevnlig alternerer mellom sekvensreferanser og ukomprimerte sekvenser på 1 kan dette dermed føre at filen blir litt større enn originalen. Vi kan forebygge dette ved å øke avstandshukommelsen til algoritmen. Vi bruker kun 1 byte for avstandshukommelsen til algoritmen vår.

Graphical user interface, text

Description automatically generated

Ved denne PDF-filen er komprimering er mindre igjen og komprimeringen fungerer til en viss grad, men den kutter kun av noen hundre bytes.

Siden vi kun bruker 1 byte som maksavstand kan det altså være en tilfeldighet at den klarte å komprimere den siste PDF-filen til en mindre størrelse og ikke den første.

Huffmann

Huffman-koding komprimerer filer ved å se på hvilke bytes som blir brukt oftest i en fil. Den bruker så et binærtre for å produsere binære koder for alle bytesene som finnes i denne filen. Huffmann-kodene er enten kortere eller like lange som den orginale byten. Dette funker derimot kun dersom antall unike bytes i filen er mindre 256. Hvis alle 256 tegn skal ha en unik huffman-kode, må huffman-treet bruke 8 ledd på å komme seg til hver eneste løvnode. Da vil alle bytes uavhengig av frekvens ha en kode som er 8 bits lang. Altså er det likegyldig om man bruker standard binærkode for bytes eller om man bruker huffman i slike tilfeller den huffmanntreet er fylt opp. Ulempen ved å velge huffman i dette tilfellet er at denne krever en frekvenstabell på ca. 1KB (vi bruker 1024kb + 32 bit, siden vi inkluderer et ekstra int i frekvenstabellen som symboliserer når algoritmen skal stoppe.) Dette vil føre til at en fil med ca. 1KB ekstra i filstørrelse.

Hvis man skal få maksimalt ut av huffmann og lempel-ziv kombinert, bør man derfor bruke huffmann på bytesekvenser som ikke kunne bli komprimert, siden disse har større sannsynlighet for å ha færre unike tegn enn hele koden tilsammmen. I et slikt tilfelle må man lage en kombinert frekvenstabellen for alle delsekvensene med ukomprimerte bytes før man starter komprimeringen. Hvis man lager en frekvenstabell for hver eneste delsekvens med ukomprimerte bytes, vil man legge til 1KB ekstra for hver delsekvens som kan føre til at man øker plassen som blir brukt isteden for å redusere den. Hvis disse delsekvensene er lange nok, kan man i teorien tjene på det til tross for at man har en egen frekvenstabell for hver delsekvens.

Siden vi skrev først Lempel-Ziv og derretter Huffman, har vi implementert det slik at vi først komprimerer med Lempel-Ziv, også komprimerer vi bytetabellen igjen denne returnerer med huffmann-koding før vi skriver den til en komprimert fil. Omvendt rekkefølge for dekomprimering. Dette funker, men kun for de mellomste filene.

De minste filene tjener ikke nok på Lempel-Ziv og deretter Huffman (i motsetning til kun Lempel-Ziv komprimering) til at det kompenserer for de ekstra 1056 bytesene vi bruker for frekvenstabellen. For de større filene får vi også en økning i størrelse på litt under 1KB ekstra med data i den komprimerte filen (sammenlignet med kun Lempel-Ziv). Det er som sagt grunnet at den bruker nesten alle 256 ulike bytes. Den mellomste filen diverse.lyx blir ca. 5KB mindre enn hvis man kun komprimerer med lempel-ziv, så her tjener man altså på å bruke begge komrimeringsalgoritmene. Alle de komprimerte filene uavhengig om de kun er komprimert med Lempel-Ziv eller Lempel-Ziv også Huffmann, er mindre enn orginalfilen. Unntaket er oppgave-pdfen. Dette som tidligere nevnt grunnet at PDF allerede er komprimert.

Vi har ikke mer kapasitet å bruke på denne innleveringen slik at vi kunne fått implementert en gunstig kombinasjon av de to komprimeringsalgoritmene istedenfor å komprimere hver for seg.

Vi har unngått å koke denne øvingen helt og har skrevet alt fra egen kunnskap + kunnskap tilegnet oss under arbeid. Dette har vært veldig krevende og dermed medført at vi har bruk svært mange timer på denne øvingen. Til gjengjeld har vi lært masse om primitive typer (spesielt bytes), bits , bit-operatorer, og ikke minst debugging.

Screenshots av Lempel-Ziv + Huffman

**.txt**

Text

Description automatically generated

**.lyx**

Text

Description automatically generated

**.pdf**

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**